THESIS

MODELING, SIMULATION, AND CONTROL OF SOFT ROBOTS

Submitted by

Ben Pawlowski

Department of Mechanical Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2019

Master's Committee:

Advisor: Jianguo Zhao

Christian Puttlitz
Charles Anderson

ABSTRACT

MODELING, SIMULATION, AND CONTROL OF SOFT ROBOTS

Soft robots are a new type of robot with deformable bodies and muscle-like actuations, which are fundamentally different from traditional robots with rigid links and motor-based actuators. Owing to their elasticity, soft robots outperform rigid ones in safety, maneuverability, and adaptability. With their advantages, many soft robots have been developed for manipulation and locomotion in recent years. However, the current state of soft robotics has significant design and development work, but lags behind in modeling and control due to the complex dynamic behavior of the soft bodies. This complexity prevents a unified dynamics model that captures the dynamic behavior, computationally-efficient algorithms to simulate the dynamics in real-time, and closed-loop control algorithms to accomplish desired dynamic responses.

In this thesis, we address the three problems of modeling, simulation, and control of soft robots. For the modeling, we establish a general modeling framework for the dynamics by integrating Cosserat theory with Hamilton's principle. Such a framework can accommodate different actuation methods (e.g., pneumatic, cable-driven, artificial muscles, etc.). To simulate the proposed models, we develop efficient numerical algorithms and implement them in C++ to simulate the dynamics of soft robots in real-time. These algorithms consider qualities of the dynamics that are typically neglected (e.g., numerical damping, group structure). Using the developed numerical algorithms, we investigate the control of soft robots with the goal of achieving real-time and closed-loop control policies. Several control approaches are tested (e.g., model predictive control, reinforcement learning) for a few key tasks: reaching various points in a soft manipulator's workspace and tracking a given trajectory. The results show that model predictive control is possible but is computationally demanding, while reinforcement learning techniques are more computationally effective but require a substantial number of training samples. The modeling, simulation, and control framework

developed in this thesis will lay a solid foundation to unleash the potential of soft robots for various applications, such as manipulation and locomotion.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Soft robotics focuses on the design, actuation, modeling, and control of soft structures for use in medical, personal, exploratory, and a potentially wide range of other applications [1–4]. These robots diverge from traditional rigid robots by being innately compliant and highly flexible. This compliance leads to several benefits over rigid robots as well as some challenges in their application. One significant benefit is greater adaptability to their environment and contacts, allowing them to take on more configurations and decrease the risk of damaging the objects they work with [2]. The primary challenge present with these robots is their complex mechanics. Being soft, they have infinite degrees of freedom allowing their dynamic behavior to be significantly more complex than traditional robots.

Soft robots have found their uses in several fields. In medical applications they have been useful for surgical devices [5], rehabilitation devices [6], and other applications [5, 7–9]. As surgical devices they can lead to less invasive surgeries [5]. For rehabilitation they support more natural motions of the body [6]. Soft robots can also mimic biological motions effectively, allowing for easier study of behavior in creatures such as fish [10]. Several biological structures have been inspiration for manipulator design such as octopus tentacles [11–14], elephant trunks [15, 16], snakes [17,18], and several others [10,19–23]. There are also benefits to using soft structures when exploring difficult environments, soft robots can be more adaptable to extreme conditions such as high heat [24] or aquatics [25] and adapting to uncertain terrain. Other applications include new takes on traditional robotics applications such as gripping [26, 27] and locomotion [19, 22, 24]. There are applications as well for reconfigurable robots which are robots that can adjust their shape by controlling the stiffness of their components [28]. This shape change allows the robot to perform different tasks simply by altering its geometry. Many applications are still in their early stages and more diverse applications are being developed.

The primary difficulty with soft robots stems from the fact that they behave as continuums rather than rigid bodies and thus their configurations have infinite degrees of freedom. While this allows for many configurations, this also makes it difficult to model and control. Modeling the dynamics of soft robots is complex due to need of continuum models of the physics [29] and that there can be complex material properties to consider. As most control schemes rely [30] on models of the dynamic behavior of the system, the computational complexity of the dynamics highly restricts the potential to control these robots. Both modeling and control are necessary for designing and applying these robots effectively. This has been a primary bottleneck for the application of soft robots.

Actuation of soft robots is also an area of significant exploration. The actuators must be soft to keep the benefits of the soft body which has led to a few primary actuation strategies. The most common approaches are to use cables/tendons [13, 31] and fluidic actuators [15]. Both are relatively simple to apply and are quite capable of actuating soft robots effectively. However, they both require rigid parts to be useful; removing some of the utility of the soft robots and adding bulk. As to cables, it is necessary to have motors and with fluidic actuators, compressors and valves. While it currently does not seem possible to have totally soft robots, there are some actuation strategies that reduce the necessary external components to just circuit boards and power supplies. These actuation strategies come under the umbrella term of artificial muscles. These are devices that generally actuate due to electrical power and include devices such as Shape Memory Alloys (SMAs) [19, 21], Dielectric Elastomer Actuators (DEAs) [23, 26, 32], Ionic Polymer-Metal Composites (IPMCs) [22, 33], and Twisted-and-Coiled actuators (TCAs) [34]. Beginning investigation has shown some useful results, but more exploration is necessary.

Modeling the behavior is a very important topic for the soft manipulators as it influences the design process and control. Generally, continuum models are rather complex inspiring the development of simplified models. The most prominent of these being the constant curvature (CC) approximation and its extensions [35]. The CC model assumes that the manipulator undergoes pure bending and the resulting curvature of the body is constant. This can be a reasonable model in the

case of the statics when external loads are relatively constant, but breaks down whenever there are significant out of plane loads (e.g., gravity and lateral contact) and dynamic behavior. Extensions to this model include building the body out of many CC sections resulting in the Piecewise Constant Curvature (PCC) [35] model which is more amenable to out of plane loads and dynamic behavior. There has been a recent further generalization as the discrete Cosserat model [36] where it assumes constant spatial twists, a more generalized deformation, so more deformation modes can be considered rather than pure bending. These approaches conveniently allow for formalisms similar to traditional rigid robots, such as DH-parameters [35] or Lagrangian models of serial manipulators [36], because it makes the constant sections akin to actuated joints in traditional robots. There have also been more traditional continuum models for the soft robots, particularly slender ones. Slender manipulators can be modelled effectively as rods or beams. Some approaches have used the elastica [37],Euler-Bernoulli beams [38], Timoshenko beams [39], Kirchhoff rods [40], and Cosserat rods [25, 31, 40]. The Cosserat rod approach is the most general of these as it models extension, shear, torsion, and bending. Other more general rod models do exist [29, 41], but a use case has not yet been found as the extra deformation modes do not appear to be significant. These continuum models are generally more accurate, but are more computationally expensive. There are also general approaches to modeling soft robots such as Finite Element Models (FEM) [42] which generalize to non-slender robots and can use standard computational software.

For control many approaches exist [30], but there has been a strong restriction in the development due to computational burden of the models. Quasi-static assumptions have been useful for developing simple starting points for control schemes despite statics not truly being a control problem [43]. For the dynamics several approaches have been used, but none have emerged as the clear best choice. These approaches typically rely on the modelled behavior and therefore either lose accuracy with simpler models or are limited by computational complexity. Due to these restrictions, machine learning approaches have been quite attractive [44]. Machine learning methods are able to produce effective control policies that are computationally efficient at the expense of potentially long training times. Considerable work has been focused on reducing the necessary

number of samples in machine learning [45–48]. One of the biggest challenges facing control is the infinite degrees of freedom in the state space. Because of this, the configuration is only partially represented, and the robot will always be inherently underactuated [49].

In this thesis we develop a general framework using Cosserat rod theory to model soft manipulators and include models for several common types of actuators and actuators gaining more interest. We also develop several real-time simulations based on Cosserat and Kirchoff rod theory. Not only are these real-time, but we have made sure the Lie group structure of the kinematics is always preserved and that no numerical damping is introduced into the simulations, which have generally not been considered for soft robot simulations. We also explore real-time control schemes for the soft manipulators. In the control we see that real-time Model Predictive Control (MPC) is difficult to achieve, but can be done in a restricted case and the Reinforcement Learning (RL) is effective for real-time control of the simulations. We will focus on the modeling of the soft manipulators via the Cosserat rod model developing the kinematics and the statics (Chapter 2), and the dynamics (Chapter 3). We will also discuss approaches to solving and simulating the behavior. After modeling, the control approaches will be discussed and explored Chapter 4. Numerical approaches for real-time control will be discussed and machine learning techniques will be presented that learn from the simulations and improve the real-time capabilities of the control schemes.

1.1 Preliminaries

Throughout this thesis ideas from Lie Theory [50], Screw Theory [51], and Variational Calculus [52] will be used. Some basic background knowledge will be presented for familiarity.

1.1.1 Lie Theory

Lie Theory [50] is a widely applicable field with lots of mathematical depth. Throughout the thesis we will be using a few key definitions presented in this section. We will focus on the Special Orthogonal group, $SO(3)$ and the Special Euclidean group, $SE(3)$, in matrix form.

$SO(3)$ is a representation of 3D rotations, $R \in \mathbb{R}^{3\times3}$, as matrices. $R \in SO(3)$ are required to have $det(R) = 1$ and $R^T R = I$. $SE(3)$ represents general rigid body transformations meaning it describes both rotations and translations, we will be representing them as homogeneous transformation matrices, $g \in \mathbb{R}^{4\times4}$, which are defined as:

$$g = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \tag{1.1}$$

where R is a rotation matrix and $p \in \mathbb{R}^3$ is the position or translation vector. Throughout the thesis $SE(3)$ elements will be referred to as configurations.

Lie groups are always differentiable and can either be left or right-invariant:

$$\text{left-invariant: } \frac{dg}{dx} = ga \tag{1.2}$$

$$\text{right-invariant: } \frac{dg}{dx} = bg \tag{1.3}$$

where a and b are different forms of the Lie algebra associated with the Lie group g and x is a parameter of the Lie group (e.g., time). We will be using the left-invariant form as the algebra elements can be interpreted as the body frame velocities. This is convenient for the development of the physics models later.

All Lie algebras are isomorphic to a vector space and we will denote the isomorphism from vectors to matrices as $\hat{}: \mathbb{R}^{n\times n} \to \mathbb{R}^p$ and the inverse as $\check{}: \mathbb{R}^p \to \mathbb{R}^{n\times n}$. The algebras for $SO(3)$ and $SE(3)$ are $so(3)$ and $se(3)$ respectively. Their representations and transformations are defined as:

$$so(3)\colon \omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \simeq \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{1.4}$$

$$se(3)\colon \xi = \begin{bmatrix} \omega \\ \nu \end{bmatrix} \simeq \begin{bmatrix} \hat{\omega} & \nu \\ 0 & 0 \end{bmatrix} \tag{1.5}$$

where $\omega \in so(3)$ is a Lie algebra element for $SO(3)$, $\xi \in se(3)$ is a Lie algebra element for $SE(3)$, and $\simeq$ means the objects are isomorphic. The form of $so(3)$ is either $\mathbb{R}^3$ or $\mathbb{R}^{3\times3}$ and the form of $se(3)$ is either $\mathbb{R}^6$ or $\mathbb{R}^{4\times4}$. Using the fact that ξ can be split into angular, ω, and linear, ν, components allows us to conveniently define the isomorphism in terms of the $so(3)$ isomorphism. It is useful to note that the matrix form of $so(3)$ are skew-symmetric matrices which make matrix-vector multiplication act like the cross product (1.6).

$$a \times b = \hat{a}b \tag{1.6}$$

For every Lie algebra we define the Lie bracket or commutator product, $[\cdot, \cdot]$, as:

$$[\hat{a}, \hat{b}] = \hat{a}\hat{b} - \hat{b}\hat{a} \tag{1.7}$$

When working with vector forms of the algebra elements we can use the adjoint representation to do the same multiplication:

$$[\hat{a}, \hat{b}] \simeq ad_a(b) \tag{1.8}$$

where ad_a is the adjoint representation of a acting on b. For $so(3)$ and $se(3)$ we can write the adjoints as matrices:

$$ad_\omega = \hat{\omega} \tag{1.9}$$

$$ad_\xi = \begin{bmatrix} \hat{\omega} & 0 \\ \hat{\nu} & \hat{\omega} \end{bmatrix} \tag{1.10}$$

There also exists the adjoint for the group defined by the group action as:

$$g\hat{a}g^{-1} \simeq Ad_g a \tag{1.11}$$

where Ad_g is the adjoint representation for the group element g. For $SO(3)$ and $SE(3)$ we can define them as:

$$Ad_R = R \tag{1.12}$$

$$Ad_g = \begin{bmatrix} R & 0 \\ \hat{p}R & R \end{bmatrix} \tag{1.13}$$

A final useful concept for Lie groups and algebras are diffeomorphisms, primarily the exponential map, which map an algebra element to a group element. Numerically the exponential map, $\exp$, is the matrix exponential. A diffeomorphism maps an algebra element to a group element local to the identity of the group, which allows for operations like:

$$g_1 = g_0 \exp(\hat{\xi}) \tag{1.14}$$

This allows us to move along the group manifold using algebra elements, which will be useful in numerical simulations. The inverse operation is the natural logarithm, $\log$, and maps group elements to algebra elements.

1.1.2 Screw Theory

Screw Theory [51] is an approach to rigid body mechanics that unifies linear and angular behavior into a single representation (e.g., moments and forces unify to a wrench). Screw Theory uses $SE(3)$ to describe rigid body motions and adds some useful terminology and concepts.

If we look at a rigid bodies' motion in time we get:

$$\dot{g} = g\hat{\eta} \tag{1.15}$$

where $\dot{}$ is the derivative with respect to time and η is a combination of the angular, ψ, and linear, κ, velocities in the body frame. This generalized velocity, η, is typically termed the twist of the body. However, we will be working with twists not only defined by a time derivative, but a space derivative when deriving the Cosserat rod model. To distinguish between them we will refer to them as temporal and spatial twists throughout.

The group action can be interpreted as a frame transformation [51]. To transform a twist from one frame to another we use:

$$\hat{\eta_2} = T\hat{\eta}_1 T^{-1} \tag{1.16}$$

where $T \in SE(3)$ is the transformation from frame 2 to frame 1. This is equivalently written as:

$$\eta_2 = Ad_T \eta_1 \tag{1.17}$$

Looking at power, P, we can define what a wrench is and use the frame invariance of power to define the wrench frame transformation. Power is defined in the typical way:

$$P = \langle \eta, W \rangle \tag{1.18}$$

where W is a wrench acting on the body and $\langle \cdot, \cdot \rangle$ is the inner product. The wrench is made up of a moment, M, and a force, F, as $W = [M, F]^T$. Using frame invariance we get the frame transformation for wrenches:

$$W_2 = Ad_T^T W_1 \tag{1.19}$$

where the superscript T stands for transpose.

1.1.3 Variational Calculus

Variational Calculus (VC) or the Calculus of Variations [52] is a mathematical technique for optimization problems and gives a way to determine the optimal trajectory function for a given objective functional. Since the physics of mechanical systems can be written as optimization problems (e.g., Hamilton's principle, Virtual Work, minimum potential energy, etc.) VC also has its uses in modeling physical systems.

The concept behind VC is that in order to find the optimal trajectory for some objective the trajectory must extremize (minimize or maximize) the given objective. This is equivalent to finding maxima and minima along curves by taking derivatives, but extends the idea from points on curves to trajectories on surfaces. The equivalent to finding where a derivative is 0 is finding where the variation of the trajectory is 0 or stationary. To illustrate the idea we present how to derive the Euler-Lagrange equations from Hamilton's principle [53].

Hamilton's principle [53] states the dynamic trajectory taken by a system is the one that extremizes the action integral. We define the action integral as the time integration of the Lagrangian.

$$S = \int_0^\tau \mathcal{L}(q, \dot{q}) dt \tag{1.20}$$

where $\mathcal{L}$ is the Lagrangian, q is the generalized coordinates for the system, $\dot{q}$ are the generalized velocities, and τ is the time horizon. With the boundary conditions $q(0)$ and $q(\tau)$ defined the dynamic system is fully specified. Hamilton's principle is then trying to find $q(t)$ that extremizes S, so to find this using VC we perturb the trajectory q and find where S is stationary.

To perturb the trajectory q we introduce a perturbing trajectory δq and a parameter ϵ and redefine q to make the action integral:

$$S(q+\epsilon\delta q)=\int_0^\tau \mathcal{L}(q+\epsilon\delta q,\dot{q}+\epsilon\dot{\delta q})dt \tag{1.21}$$

We need to enforce $\delta q(0) = \delta q(\tau) = 0$ so that boundary conditions are respected for the original problem.

To find the stationary point of S we are looking for the trajectory where a small perturbation does not change S which can be stated as:

$$\frac{d}{d\epsilon}S(q+\epsilon\delta q)|_{\epsilon=0}= 0 \tag{1.22}$$

showing the similarities to finding minima and maxima using derivatives.

Applying this to the action integral we get:

$$\int_0^\tau (\frac{\partial\mathcal{L}}{\partial q}\delta q+\frac{\partial\mathcal{L}}{\partial\dot{q}}\dot{\delta q})dt=0 \tag{1.23}$$

We want to get rid of $\dot{\delta q}$ so that we only have δq terms. This can be done with integration by parts:

$$\frac{\partial\mathcal{L}}{\partial\dot{q}}\delta q|_0^\tau+\int_0^\tau \delta q(\frac{\partial\mathcal{L}}{\partial q}-\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{q}})dt=0 \tag{1.24}$$

where the first term will evaluate to 0 since we define $\delta q(0) = \delta q(\tau) = 0$. Knowing that δq must be arbitrary in time and that the integral must evaluate to 0 we know that the inner term must be 0:

$$\frac{\partial\mathcal{L}}{\partial q}-\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{q}}=0 \tag{1.25}$$

these are the Euler-Lagrange equations derived from Hamilton's principle.

We see that, once the objective is defined, using VC is a straightforward process to follow. We can define the first variation, equivalent to the first derivative, as:

$$\delta(f(x))=\frac{d}{d\epsilon}f(x+\epsilon\delta x)|_{\epsilon=0} \tag{1.26}$$

where we use $\delta()$ to mean the first variation and prepending a δ to a trajectory indicates the variation of the trajectory. It is occasionally necessary to use higher order variations, but in this thesis only the first variation is necessary.

There are a few special results when using VC on Lie groups and algebras. The variation of a group element is defined as:

$$\delta(g) = \frac{d}{d\epsilon} g \exp(\epsilon \hat{\delta h})|_{\epsilon=0} = g\hat{\delta h} \tag{1.27}$$

This can then be used to define the variation of the algebra following the steps:

$$\delta(\dot{g}) = \delta(g\hat{\eta}) \tag{1.28}$$

$$\frac{d}{dt}(g\hat{\delta h}) = g\hat{\delta h}\hat{\eta} + g\hat{\delta\eta} \tag{1.29}$$

$$g\hat{\eta}\hat{\delta h} + g\frac{d}{dt}\hat{\delta h} = g(\hat{\delta h}\hat{\eta} + \hat{\delta\eta}) \tag{1.30}$$

$$\hat{\delta\eta} = \frac{d}{dt}\hat{\delta h} + [\hat{\eta}, \hat{\delta h}] \tag{1.31}$$

$$\delta\eta = \frac{d}{dt}\delta h + ad_\eta \delta h \tag{1.32}$$

where (1.32) is the vector form of the variation for a Lie algebra.

Chapter 2

Soft Manipulator Kinematics and Statics

Modeling soft manipulators requires general continuum models of the bodies, but given that the majority of soft manipulators are slender, much longer than they are thick. This allows us to simplify from general continuum models to rod models. The main assumption for a rod model is that the body has only one major spatial dimension along its axis. We know that the manipulators will undergo significant body deformations, so the modeling approach will need to handle these large deformations. One such approach is the Cosserat rod theory [25, 31, 54], where we are able to model bending, torsion, shear, and extension. There exist other models for rods that incorporate more [41] or less [39] modes of deformation, but the Cosserat model seems to sit at a good middle ground for soft robotics.

The idea behind a Cosserat model is to assume that the body is made of infinitesimal rigid bodies rather than point particles [55]. Rigid bodies means the configuration of the body is described by a position and orientation rather than just a position. For the rod this means the cross sections are the rigid bodies and are considered rigid planes. Cosserat models can also be referred to as micropolar theories as the microstructure [55] has both position and orientation. The other important idea is the directed curve. This is the curve passing through the centroid of all the cross sections of a rod, the centerline, and at every point along the curve there exists an orthonormal frame, the directors. The directors act as the body frame for a cross section and allow us to relate the position and orientation to the fixed frame. Note that the directors do not need to be orthonormal, if allowed to vary more deformations of the microstructures can be modelled [41], but the added complexity is not worth it in this case. The modeling will begin with the kinematics of the rods using $SE(3)$ to describe the configuration of the body and then derive the statics followed by the dynamics including the relevant phenomena (e.g., inertia, strain, gravity, etc.).

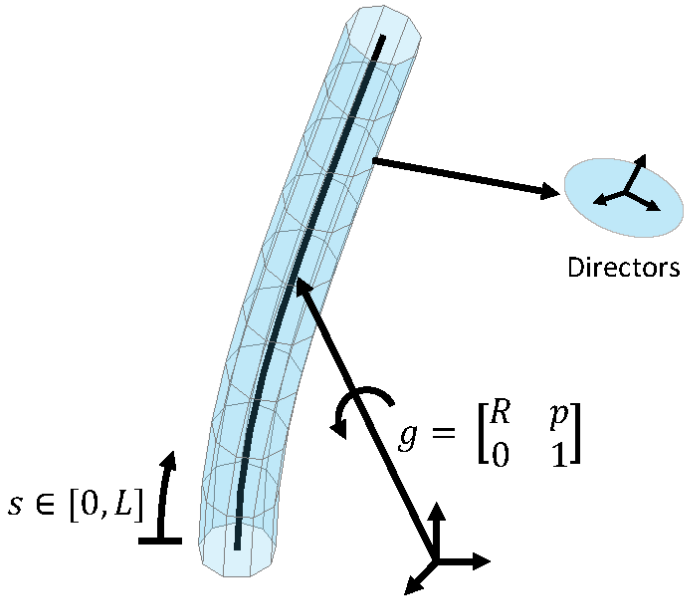


Figure 2.1: A diagram for the kinematics of the Cosserat rod. Shows the centerline $s \in [0, L]$ as the black curve, the configuration g, the directors at a cross section, and the body of the rod in blue.

2.1 Kinematics

First we define the centerline, $s \in [0, L]$, as the curve passing through the centroids of all the cross sections where L is the length of the rod in its reference configuration. At every point s along the centerline there exists the a set of directors. The z axis of the directors is tangent to s and the x and y axes are assumed to be aligned with the principal axes of the cross sections. For each set of directors we have a rotation and a translation relative to the fixed frame. We can describe the rotation and translation by a homogeneous transformation matrix $g \in SE(3)$ (1.1) as discussed in the preliminaries. We always define the fixed frame to be at the base of the manipulator so that $g(0) = I_{4\times4}$. Repeating the definition of g (1.1):

$$g = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

where $R \in SO(3)$ is a rotation matrix and $p \in \mathbb{R}^3$ is the position vector. g thus gives a description of the position and orientation of all the points along the centerline and the configuration for the cross sections.

In order to determine strains and deformations for any point in the body we need to describe the kinematics of every point. With the centerline and its kinematics, g, already defined we can locate any other point in the body as a translation, $T \in SE(3)$, from the centerline. Here is where we start using the rigid cross section assumption. By having the cross sections be rigid, any point in the cross section is just a displacement, r, from the centerline.

$$G(s, x, y) = g(s)T(x, y) \tag{2.2}$$

$$T(x, y) = \begin{bmatrix} I_{3\times3} & r(x, y) \\ 0 & 1 \end{bmatrix} \tag{2.3}$$

$$r(x, y) = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \tag{2.4}$$

where $G \in SE(3)$ is the configuration of any point in the body, $x, y \in \mathbb{R}$ denote the position in the cross section along the principle axes, and $I_{3\times3}$ is the identity matrix. It is possible to have a more complex T to accommodate structural features of the material or body, such as oriented fibers, but in our case this is unnecessary.

Due to the centerline being defined as passing through the centroid of all the cross sections and the displacement r being defined relative to the centroid we know that:

$$\int r dA = 0 \tag{2.5}$$

due to symmetry. This will be useful in later derivations.

For the centerline we can define the spatial twist, $\xi \in \mathbb{R}^6$, as:

$$\frac{\partial g}{\partial s} = g' = g\hat{\xi} = g \begin{bmatrix} \hat{\omega} & \nu \\ 0 & 0 \end{bmatrix} \tag{2.6}$$

where $'$ denotes differentiation with respect to s, $\omega \in \mathbb{R}^3$ is the angular component of ξ, and $\nu \in \mathbb{R}^3$ is the linear component. ξ can be viewed as the relative configuration change between adjacent cross sections along the centerline. Later, ξ will be related to the strain and deformation in the body and the change in ξ from its reference value ξ^* will act as a measure of strain. ξ^* comes from the initial configuration of the rod and the most common case will be the rod is initially straight giving $\xi^* = [0, 0, 0, 0, 0, 1]^T$. Other initial configurations (e.g., initially bent) can be defined as well. Throughout we will refer to values in the reference configuration with a superscript $*$.

Later, we will be computing the strain energy and will need the definition for the deformation gradient, $F \in \mathbb{R}^{3x3}$, to determine the deformations and strains [54]. The components of F are $\frac{\partial p_G}{\partial x_i}$ where $x_i = x, y, s$ and p_G is the position component of G.

$$p_G = Rr + p \tag{2.7}$$

$$\frac{\partial p_G}{\partial x} = R\vec{x} \tag{2.8}$$

$$\frac{\partial p_G}{\partial y} = R\vec{y} \tag{2.9}$$

$$\frac{\partial p_G}{\partial s} = R(\nu + \hat{\omega}r) \tag{2.10}$$

$$F = R \begin{bmatrix} \vec{x} & \vec{y} & \nu - \hat{r}\omega \end{bmatrix} \tag{2.11}$$

where $\vec{x} = [1, 0, 0]^T$ and $\vec{y} = [0, 1, 0]^T$ are the unit vectors along the principal axes of the cross sections. To simplify the notation some we define:

$$\begin{aligned}\Gamma &= \nu - \hat{r}\omega \\ &= \Delta\nu + \nu^* - \hat{r}(\Delta\omega + \omega^*) \\ &= \gamma + \gamma^*\end{aligned} \tag{2.12}$$

$$\gamma = \Delta\nu - \hat{r}\Delta\omega, \gamma^* = \nu^* - \hat{r}\omega^* \tag{2.13}$$

which makes $F = R[\vec{x}, \vec{y}, \Gamma]$.

To measure the strains we will use the Green-Lagrange strain tensor [54], $\varepsilon \in \mathbb{R}^{3x3}$. The Green-Lagrange strain is a measure of finite strain, rather than infinitesimal strain, which is important for cases of large deformations like in soft robots and is posed in the Lagrangian viewpoint of the coordinates which is compatible with Hamilton's principle used in the later derivations. The Green-Lagrange strain tensor is defined as $\varepsilon = \frac{1}{2}(F^T F - (F^*)^T F^*)$, or in component form $\varepsilon_{ij} = \frac{1}{2}(\langle \frac{\partial p_G}{\partial x_i}, \frac{\partial p_G}{\partial x_j} \rangle - \langle \frac{\partial p_G^*}{\partial x_i}, \frac{\partial p_G^*}{\partial x_j} \rangle)$.

$$\begin{aligned}\varepsilon_{sx} &= \frac{1}{2}\langle \gamma, \vec{x} \rangle \\ \varepsilon_{sy} &= \frac{1}{2}\langle \gamma, \vec{y} \rangle \\ \varepsilon_{ss} &= \frac{1}{2}\langle \gamma, \gamma \rangle + \langle \gamma, \gamma^* \rangle \\ \varepsilon_{xx} &= \varepsilon_{xy} = \varepsilon_{yy} = 0\end{aligned} \tag{2.14}$$

This defines the relevant kinematics dependent on the shape of the manipulator.

For the time dependence we define the temporal twist, or generalized velocity, as:

$$\frac{\partial g}{\partial t} = \dot{g} = g\hat{\eta} = g \begin{bmatrix} \hat{\psi} & \kappa \\ 0 & 0 \end{bmatrix} \tag{2.15}$$

where $\dot{}$ is the derivative with respect to time, $\eta \in \mathbb{R}^6$ is the temporal twist, $\psi \in \mathbb{R}^3$ is the angular velocity, and $\kappa \in \mathbb{R}^3$ is the linear velocity. The temporal twist defines the velocities of the cross sections in the body frame.

Using the symmetry of second order partial derivatives ($\frac{\partial}{\partial x}\frac{\partial}{\partial y} = \frac{\partial}{\partial y}\frac{\partial}{\partial x}$) and computing the partials of g we can also establish the relationship between twists:

$$\dot{\xi} = \eta' + ad_{\xi}\eta \tag{2.16}$$

This summarizes the important quantities in the kinematics: g, ξ, η, F, and ε. For the behavior of the manipulators we develop the statics in the following section and the dynamics in the following chapter.

2.2 Statics

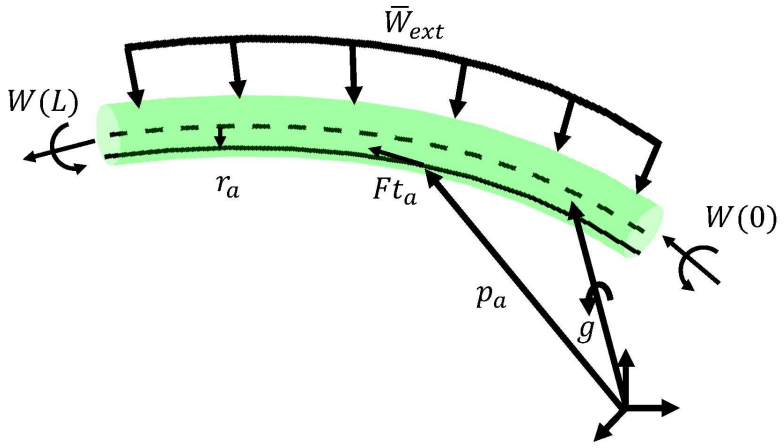


Figure 2.2: A diagram for the statics of the Cosserat rod. Shows the rod (green) responding to a distributed external load, $\bar{W}_{ext}$ and a cable actuator embedded in the body.

To develop the statics we want to balance the internal loads due to deformation with the external loads due to actuation, gravity, etc. We can use the principle of Virtual Work using VC to derive the static balance between the internal strain energy, U, and the other sources of work, W. The

virtual work is written:

$$\delta(W) - \delta(U) = 0 \tag{2.17}$$

where $U \in \mathbb{R}$ is the strain energy, $W \in \mathbb{R}$ is the applied work, and $\delta()$ is the first variation (1.26). For the work we can define the variation as:

$$\delta(W) = \int_0^L \langle \delta h, \bar{W} \rangle ds \tag{2.18}$$

where $\delta h \in \mathbb{R}^6$ is the variation of the configuration and $\bar{W} \in \mathbb{R}^6$ is the distributed wrench applied to the body. Here we are using h to stand for the vector form of g which is comprised of 3 angles and 3 positions. Note that in the principle of Virtual Work the loads are not varied only the displacements are hence why we only have δh. This is convenient for formulating arbitrary loads as they can be included into $\bar{W}$ without having to redo the calculations.

To finish Virtual Work we need to define the strain energy, U, and determine its variation.

2.3 Strain Energy

The internal strain energy for a body is defined as the integral of the stress times times the strain over the whole body [54] giving:

$$U = \frac{1}{2} \int \sum_{i,j \in \{x,y,s\}} \sigma_{ij} \varepsilon_{ij} dV \tag{2.19}$$

where σ is the second Piola-Kirchhoff stress tensor [54] since it is conjugate tot the Green-Lagrange strain. From the prior calculations of ε (2.14) the expression can be reduced to:

$$U = \int (\sigma_{ss}\varepsilon_{ss} + \sigma_{sx}\varepsilon_{sx} + \sigma_{sy}\varepsilon_{sy}) dV \tag{2.20}$$

Expanding and rearranging the terms inside the integral we get:

$$
\begin{aligned}
U = \frac{1}{2}(& \int (\langle \Delta\nu, \int (\sigma_{ss}(\gamma + 2\gamma^*) + \sigma_{sx}\vec{x} + \sigma_{sy}\vec{y})dA \rangle ds) \\
& + \int (\langle \Delta\omega, \int (\hat{r}(\sigma_{ss}(\gamma + 2\gamma^*) + \sigma_{sx}\vec{x} + \sigma_{sy}\vec{y}))dA \rangle ds))
\end{aligned}
\tag{2.21}
$$

which can be represented more compactly as:

$$
U = \frac{1}{2} \int \langle \Delta\xi, \Psi(\xi) \rangle ds \tag{2.22}
$$

where Ψ is the internal wrench due to the strain in the body and $\Delta\xi = \xi - \xi^*$ is the change in ξ. $\Delta\xi$ will also be referred to as the strain in the rod due to its relationship to the strain energy.

To evaluate the integral and determine Ψ we need to define σ. σ is determined from the constitutive law for the body material. It is desirable to use a simple constitutive model for the sake of efficient simulations. Also, due to the rigid plane assumption some material properties do not make sense, like Poisson's Ratio, leading to a simpler model as well. In the simplest model of the material behavior we have the linear relationships:

$$
\sigma_{ss} = E\varepsilon_{ss} \tag{2.23}
$$

$$
\sigma_{sx} = \frac{1}{2}G\varepsilon_{sx} \tag{2.24}
$$

$$
\sigma_{sy} = \frac{1}{2}G\varepsilon_{sy} \tag{2.25}
$$

$$
\sigma_{xx} = \sigma_{yy} = \sigma_{xy} = 0 \tag{2.26}
$$

where E is the Young's modulus of the body, G is the shear modulus, and the $\frac{1}{2}$ factors are to translate to the proper shear strains from the Green-Lagrange strains.

We can substitute in these definitions to determine the form for $\Phi(\xi)$; however, if we assume that the second-order terms in the Green-Lagrange strain tensor are negligible ($\langle \gamma, \gamma \rangle \approx 0$) we get a simple matrix form for Ψ. Assuming an initially straight configuration, $\gamma^* = [0, 0, 1]^T$, the quadratic terms are negligible, and that the constitutive law is linear elastic we get:

$$
\begin{aligned}
U = &\frac{1}{2}\int \langle \Delta\nu, EA\Delta\nu_z + GA\Delta\nu_y + GA\Delta\nu_x\rangle ds \\
&+ \frac{1}{2}\int \langle \Delta\omega, GJ\Delta\omega_z + EI_y\Delta\omega_y + EI_x\Delta\omega_x\rangle ds
\end{aligned}
\tag{2.27}
$$

where A is the cross sectional area, I_x and I_y are the second moments of area for the cross section, and J is the polar second moment of area for the cross section. This used the centroid definition (2.5) for some simplifications. This can be simplified further to:

$$
U = \int \frac{1}{2}\langle \Delta\xi, K\Delta\xi\rangle ds \tag{2.28}
$$

where $K = textdiag([EI_x, EI_y, GJ, GA, GA, EA])$ is the stiffness matrix for the rod. This provides a simple and reasonably accurate form of the strain energy and is computationally efficient [56].

The variation of the strain energy can be found using the variation of ξ. The variation of ξ follows from (1.32) and is:

$$
\delta\xi = \delta h' + ad_\xi \delta h \tag{2.29}
$$

The variation of the strain energy, U, is then:

$$
\delta(U) = \int_0^L \langle \delta h' + ad_\xi \delta h, K\Delta\xi\rangle ds = \langle \delta h, K\Delta\xi\rangle|_0^L + \int_0^L \langle \delta h, ad_\xi^T K\Delta\xi - (K\Delta\xi)'\rangle ds \tag{2.30}
$$

where integration by parts was used to get the rightmost representation.

Putting together (2.18) and (2.30) the statics can be written as:

$$
-\langle \delta h, K\Delta\xi\rangle|_0^L + \int_0^L \langle \delta h, \bar{W} - ad_\xi^T K\Delta\xi + (K\Delta\xi)'\rangle ds = 0 \tag{2.31}
$$

which is the weak form of the statics. To get the strong form notice that (2.31) must hold for arbitrary variations thus the strong form is:

$$\bar{W} - ad_{\xi}^{T} K\Delta\xi + (K\Delta\xi)' = 0 \tag{2.32}$$

The full system of equations for the statics is (2.6), (2.32), and the relevant boundary conditions.

There are two basic types of boundary conditions: fixed and free. In the fixed case g is known at the end, this makes δh at that end 0 in the weak form. For the free end we have $K\Delta\xi = W_{ext}$ where W_{ext} is the applied point wrench at the end, so in the case of no applied wrench the boundary condition is $\xi = \xi^*$.

For the considered soft manipulators the system will typically have the base fixed and the tip free which gives the following system of ODEs for the statics:

$$g' = g\hat{\xi} \tag{2.33}$$

$$\bar{W} - ad_{\xi}^{T} K\Delta\xi + (K\Delta\xi)' = 0 \tag{2.34}$$

with the boundary conditions of $g(0) = I_{4\times4}$ and $K\Delta\xi(L) = W_{ext}$.

2.4 External Loading

All external loadings are components of $\bar{W}$. This includes effects like actuator loading and gravity for the statics. In the formulation used, all distributed external loads have to be defined in the body frame. For some loads this is straightforward to determine and for others it is easier to transfer a fixed frame formulation to the body frame with:

$$\bar{W}_{body} = Ad_{g}^{T} \bar{W}_{fixed} \tag{2.35}$$

Some common and useful loads for the statics will be derived in the following subsections.

2.4.1 Gravity

The gravity formulation is fairly straightforward. The distributed load being applied at a given cross section is the mass of the cross section, ρA, multiplied by the gravitational acceleration in

the body frame, $R^T g_r$, where ρ is the density of the body material and g_r is the gravitational acceleration vector. Putting this together the distributed wrench due to gravity is:

$$\bar{W}_{grav} = \rho A \begin{bmatrix} 0 \\ R^T g_r \end{bmatrix} \tag{2.36}$$

If the manipulator is operated in a fluid, buoyancy can be included by making the substitution $\rho = \rho_{body} - \rho_{fluid}$.

2.4.2 Cable-Driven

Actuators that only exert axial loads (e.g., tension in cables) have the same general form for the distributed wrench. First, begin by defining the wrench exerted by the actuator in the fixed frame.

$$W_{act,fixed} = F \begin{bmatrix} \hat{p}_a t_a \\ t_a \end{bmatrix} \tag{2.37}$$

where F is the magnitude of the force, $p_a = Rr_a + p$ is the position of the actuator with r_a the displacement of the actuator from the centerline, and $t_a = \frac{p'_a}{\|p'_a\|}$ is the unit tangent vector to the actuator. To get the distributed wrench take the derivative of (2.37) with respect to s.

$$\bar{W}_{act,fixed} = F \begin{bmatrix} \hat{p}'_a t_a + \hat{p}_a t'_a \\ t'_a \end{bmatrix} + F' \begin{bmatrix} \hat{p}_a t_a \\ t_a \end{bmatrix} \tag{2.38}$$

$$p'_a = R(\nu - \hat{r}_a \omega + r'_a) \tag{2.39}$$

$$p''_a = R(\hat{\omega}(\nu - \hat{r}_a \omega + r'_a) + \nu' - \hat{r}_a \omega' - \hat{r}'_a \omega + r''_a) \tag{2.40}$$

$$t_a = \frac{p'_a}{\|p'_a\|} \tag{2.41}$$

$$t'_a = -\frac{(\hat{p}'_a)^2}{\|p'_a\|^3} p''_a \tag{2.42}$$

the derivation for t'_a is found in Appendix A. Note that $\hat{p}'_a t_a = 0$ due to $\hat{x}x = 0$ (1.6). This simplifies to:

$$\bar{W}_{act,fixed} = F \begin{bmatrix} \hat{p}_a t'_a \\ t'_a \end{bmatrix} + F' \begin{bmatrix} \hat{p}_a t_a \\ t_a \end{bmatrix} \tag{2.43}$$

To get back into the body frame we use the frame transformation (2.35).

$$\bar{W}_{act} = \begin{bmatrix} R^T & -R^T\hat{p} \\ 0 & R^T \end{bmatrix} (F \begin{bmatrix} \hat{p}_a t'_a \\ t'_a \end{bmatrix} + F' \begin{bmatrix} \hat{p}_a t_a \\ t_a \end{bmatrix}) \tag{2.44}$$

which becomes:

$$\bar{W}_{act} = F \begin{bmatrix} \hat{r} R^T t'_a \\ R^T t'_a \end{bmatrix} + \frac{F'}{\|\nu - \hat{r}_a\omega + r'_a\|} \begin{bmatrix} -\hat{p}(\nu - \hat{r}_a\omega + r'_a) \\ \nu - \hat{r}_a\omega + r'_a \end{bmatrix} \tag{2.45}$$

So, for cable like actuators we have (2.45) to define its distributed wrench. When there are multiple cables the distributed wrenches can simply be summed together and if they only run through part of the body then the load is piecewise over the relevant part of the body.

For a frictionless cable $F = -T$ where T is the tension, which can easily be substituted into (2.45).

In the case of a cable, it will always be fixed to some point in the body, typically the end, and it will exert a point wrench due to the tension. This can be written in the body frame as:

$$W_{act,point} = F \begin{bmatrix} \hat{r}_a R^T t_a \\ R^T t_a \end{bmatrix} \tag{2.46}$$

For the cables fixed at the end this contributes to the tip loading $K\Delta\xi(L) = W_{ext}$. If the cables only run through part of the body then the point loads can either be modelled as Dirac delta distributions or the rod can be split into several rods in series.

If we were to apply this to a control scheme we would be considering the tensions to be the inputs or control variables, but it is also possible to switch to using the displacement of the cables as the control inputs. To measure the displacement of the cable we use:

$$\Delta L = \int_0^L \|p_a'\| ds - \int_0^L \|(p_a^*)'\| ds \tag{2.47}$$

where ΔL is the displacement of the cable and this is the difference in the arc length in the current configuration and the original configuration.

2.4.3 Fluidic Actuators

Frequently, soft manipulators make use of fluidic actuators where a fluid is used to inflate an internal chamber. Here we demonstrate an approach to modeling them. Some simplifying assumptions about the deformations will have to be assumed to maintain the rigid cross section assumption.

First, the position of the centerline of the chamber is the same as with the tendon actuator, $p_a = Rr_a + p$. For the surface of the chamber the position can be described as:

$$p_c = R(r_c \begin{bmatrix} cos(\theta) \\ sin(\theta) \\ 0 \end{bmatrix} + r_a) + p \tag{2.48}$$

where r_c is the radius of the fluidic chamber and θ is the angle around the center of the chamber. To evaluate the distributed wrench in the fixed frame we can formulate it as the integral of the pressure exerted along a section of the surface in the direction normal to the surface.

$$\bar{W}_{act,fixed} = \int_0^{2\pi} Pr_c \begin{bmatrix} \hat{p}_c \vec{n} \\ \vec{n} \end{bmatrix} d\theta \tag{2.49}$$

where P is the pressure in the chamber and $\vec{n} = [cos(\theta), sin(\theta), 0]^T$ is the unit normal to the surface. Evaluating this results in:

$$\bar{W}_{act,fixed} = -\pi P r_c^2 \begin{bmatrix} Rd \\ 0 \end{bmatrix} \tag{2.50}$$

where d is a vector dependent on how the rotation, R, is parameterized. For example, if R is the rotation matrix using xyz-Euler angles then $d = [\cos(\theta_y)\sin(\theta_x), \sin(\theta_y), \cos(\theta_x)\sin(\theta_z) + \cos(\theta_y)\sin(\theta_z) + \cos(\theta_z)\sin(\theta_x)\sin(\theta_y)]^T$. Transforming this into the body frame gives:

$$\bar{W}_{act} = -\pi P r_c^2 \begin{bmatrix} d \\ 0 \end{bmatrix} \tag{2.51}$$

defining the distributed wrench for the fluidic actuator. So there are no forces as there is nothing to push against, but there is a moment that restores the cylinder back to being straight. Often the moment can be neglected and there is no distributed load along the body.

When using a fluidic actuator there will always be a cap somewhere in the chamber, typically at the end, and this will exert a point wrench due to the pressure acting on the cap. This can be written in the body frame as:

$$W_{act,point} = P\pi r_c^2 \begin{bmatrix} \hat{r}_a R^T t_a \\ R^T t_a \end{bmatrix} \tag{2.52}$$

where t_a is the same as defined for the cable-like actuator.

Throughout, it was assumed that the pressure, P, is controlled; however, sometimes the volume of the fluid in the chamber is controlled instead. The volume of the chamber is:

$$V_c = \int_0^L \pi r_c^2 ds \tag{2.53}$$

For simulating it is still necessary to determine the pressure as a function of the volume, which will depend on the fluid being used. The combination of V_c and $P(V_c)$ can be seen as algebraic

constraints, or in a numerical solution as boundary conditions on P when V_c is treated as a state variable.

2.4.4 Artificial Muscles

There has been development for other types of actuators that have potential to overcome the weaknesses of cables and pneumatics. These come under the general term artificial muscles and includes TCAs [34], SMAs [21], IPMCs [18], and DEAs [26]. Artificial muscles generally operate by turning applied electrical power into deformation. These are then embedded within the bodies of the soft manipulators. Artificial muscles can have some impressive properties, for example TCAs can typically perform comparably to human muscle fibers in both deformation and strength [57]. One major benefit is they only require a power supply and a control circuit reducing the need for extra rigid components. When using the artificial muscles they are embedded into the soft material and there are several ways to do this. The most straightforward case is to embed the muscle during the manufacturing of the soft body such that the muscle and the body are fully coupled, the motion of the muscle and the body are linked together. The other option is to leave space in the body for the muscle to be inserted into and then fixed at the ends. In this case the motions are not completely coupled as the muscle has some freedom to move inside the body. It is also possible to have the actuators external to the body, but we will not consider that case as it usually is not too useful as the muscles need some structural support from the body. When a muscle is fully embedded the coupling generally will not cause an issue with the numerical solution; however, in the case when the actuator is inserted into the body the global deformations are coupled and this can cause difficulties in the numerical solution process. When the artificial muscles' performance depends on a global deformation a loop develops where it is necessary to know the actuator deformation to determine the body deformation, but it is necessary to know the body deformation to determine the actuator deformation. We will demonstrate one way of dealing with this numerically.

To keep the model fairly general we will treat the artificial muscles as rods that generate a load. The load is then dependent on a generalized displacement, Δ, the spatial twist in the actuator, ξ_a,

and an input q. Δ will represent a global deformation of the muscle, some overall deformation of the actuator, while ξ_a will be related to local deformations or strains in the muscles. To start we can write the location of the muscle relative to the body with:

$$g_a = gT, T = \begin{bmatrix} R_a & r_a \\ 0 & 1 \end{bmatrix} \tag{2.54}$$

where g_a is the configuration of the muscle and T is the relative displacement from g to g_a with R_a as the relative orientation and r_a the relative translation from the centroid of the cross sections. This is similar to how the general kinematics was setup (2.2), but allows for arbitrary orientations and is only locating a single point.

We can then define the spatial twist in the muscle as:

$$g_a' = g_a \hat{\xi}_a \tag{2.55}$$

where ξ_a can be determined from (2.54) by:

$$g_a' = g\hat{\xi}T + gT\hat{\xi}_T \tag{2.56}$$

$$g_a\hat{\xi}_a = g(\hat{\xi}T + T\hat{\xi}_T) \tag{2.57}$$

$$\hat{\xi}_a = T^{-1}\hat{\xi}T + \hat{\xi}_T \tag{2.58}$$

$$\xi_a = Ad_{T^{-1}}\xi + \xi_T \tag{2.59}$$

where ξ_T is the spatial twist for the displacement T and is defined by how the muscle is embedded into the body. If T is simply a constant displacement then $\xi_T = 0$. With ξ and ξ_T known the spatial twist of the muscle is determined.

It is also necessary to know the derivative of ξ_a for the stiffness of the muscles. We may determine that as:

$$T\hat{\xi}_a = \hat{\xi}T + T\hat{\xi}_T \tag{2.60}$$

$$T\hat{\xi}_T\hat{\xi}_a + T\hat{\xi}'_a = \hat{\xi}'T + \hat{\xi}T\hat{\xi}_T + T\hat{\xi}_T\hat{\xi}_T + T\hat{\xi}'_T \tag{2.61}$$

$$\hat{\xi}'_a = T^{-1}\xi'T + T^{-1}\hat{\xi}T\hat{\xi}_T + \hat{\xi}_T\hat{\xi}_T + \hat{\xi}'_T - \hat{\xi}_T\hat{\xi}_a \tag{2.62}$$

which does not happen have a nice simplified form.

To model the physics of the artificial muscle we need to include the muscles' stiffness and its generated load, so we use the Cosserat rod approach again.

$$\bar{W}_{gen}(q, \xi_a, \Delta) + \bar{W}_{body} - ad^T_{\xi_a} K_a \Delta\xi_a + (K_a\Delta\xi_a)' = 0 \tag{2.63}$$

where $\bar{W}_{gen}$ is the generated distributed wrench, $\bar{W}_{body}$ is the distributed wrench due to the body of the manipulator, and q is the input to the muscle (e.g., voltage). Using a frame transformation and substituting for $\bar{W}_{body} = \bar{W}$ in the body model we get:

$$-ad^T_\xi K\Delta\xi + (K\Delta\xi)' + Ad^T_T(ad^T_{\xi_a} K_a\Delta\xi_a - (K_a\Delta\xi_a)' - \bar{W}_{gen}) + \bar{W} = 0 \tag{2.64}$$

If the generated distributed wrench is not dependent on a global deformation then this can be integrated similarly to the regular statics case, it is just necessary to determine ξ_a and ξ'_a from ξ and ξ'. In this situation all the deformation information is local.

If there is a dependence on the global deformation, Δ, then we have to work to uncouple the problem so that it can be integrated. The issue arises because the actuator load needs to know the global displacement, but we don't know the global displacement until the system is integrated and we cannot integrate without the actuator loads. Generally the global deformation, Δ, will take on a form similar to:

$$\Delta = \int_0^L f(g, \xi, s)ds \tag{2.65}$$

where f is some function for measuring the deformation. For example, if we need the total length of the actuator we can integrate to find its arclength as:

$$\Delta = \int_0^L \|p_a'\| ds = \int_0^L \|\nu - \hat{r}_a \omega\| ds \tag{2.66}$$

We can see that in order to determine Δ we need to know the configuration and spatial twist over the entire body, but to determine that we need to know the generated load which requires knowing Δ.

To deal with the coupling we add boundary conditions and ODEs to the system of existing equations. Since, we need a Δ to perform the integration we provide a guess, Δ_{guess}, that is used throughout the integration process. Then we need to make sure that the guess was correct, so we add the boundary condition that the guess and the integrated value must match, $\Delta_{guess} = \Delta_{int}$. To integrate we get the ODE for Δ and add it to the system of equations.

$$\Delta' = f(g, \xi, s) \tag{2.67}$$

Essentially the process adds another state variable Δ to the system of equations, and unknown initial condition Δ_{guess}, an ODE $\Delta' = f(g, \xi, s)$, and a boundary condition $\Delta = \Delta_{guess}$. This fixes the coupling issue by providing a Δ to be used in computing the actuator loads during integration and sets up conditions to make sure the right Δ is used in the computations via the boundary condition and ODE. The downside to this would typically be that the system is now a boundary value problem (BVP) which is a harder to solve than an initial value problem (IVP), but the statics already was a BVP due to mixed boundary conditions, so very little computational complexity is incurred.

2.5 Constant Spatial Twist Solutions

The statics generally does not have an analytic solution; however, given that it is common and useful to assume a constant ξ [35, 36] we can see what is the required loads for the assumption to hold. Assuming that ξ is constant and substituting into the statics (2.32) we get:

$$\bar{W} = ad_{\xi}^{T} K \Delta\xi \tag{2.68}$$

A constant ξ therefore implies that the distributed wrench must be constant as well.

When the ends are free and ξ is constant then the applied wrenches at the ends must be:

$$W_{ext} = K\Delta\xi \tag{2.69}$$

If an end is fixed it doesn't influence the solution, though if both ends are fixed then the tip and base configuration, $g(L)$ and $g(0)$, need to be consistent with the constant ξ definition. Unfortunately there appears to be no analytic solution for g even in the constant ξ case. For example consider the assumed solution:

$$g(s) = g_0 \exp(\theta(s)), \theta(0) = 0 \tag{2.70}$$

where $\theta(s)$ is a dummy variable that needs to be solved for. Taking the derivative we get:

$$g' = g_0 \exp(\theta(s)) \operatorname{dexp}_{-\theta(s)} \theta' = g_0 \exp\theta(s))\hat{\xi} \tag{2.71}$$

where dexp is the differential of the exponential map. For $SE(3)$ dexp has an analytic form [58].

$$\mathrm{dexp}_\xi = I + a(\theta) ad_\xi + c(\theta) ad_\xi^2 + (\omega^T \nu) \begin{bmatrix} 0 & 0 \\ Q(\omega) & 0 \end{bmatrix} \tag{2.72}$$

$$\theta = \|\omega\| \tag{2.73}$$

$$a(\theta) = \frac{\sin(\theta)}{\theta} \tag{2.74}$$

$$b(\theta) = \frac{1 - \cos(\theta)}{\theta^2} \tag{2.75}$$

$$c(\theta) = \frac{1 - a(\theta)}{\theta^2} \tag{2.76}$$

$$Q(\omega) = \frac{a(\theta) - 2b(\theta)}{\theta^2} \hat{\omega} + \frac{b(\theta) - 3c(\theta)}{\theta^2} \hat{\omega}^2 \tag{2.77}$$

Solving $\mathrm{dexp}_{-\theta(s)}\, \theta' = \xi$ does not appear to have an analytic solution. However, the first and second order Taylor expansions of dexp both give:

$$g = g_0 \exp(\xi s) \tag{2.78}$$

which is a fairly simple and convenient form. Higher order approximations do not seem obvious to solve for the general case, but the low order approximation is a decent starting point.

To give a more concrete form to this assume that the rod is initially straight and that it is a cylinder, we have $\xi^* = [0, 0, 0, 0, 0, 1]^T$ and $K = \mathrm{diag}([EI, EI, GJ, GA, GA, EA])$ with I, J, and A being the relevant definitions for a circular cross section. For a constant ξ solution we have:

$$\bar{W} = \begin{bmatrix} GA\nu_y\nu_z - EA\nu_y(\nu_z - 1) + EI\omega_y\omega_z - GJ\omega_y\omega_z \\ EA\nu_x(\nu_z - 1) - GA\nu_x\nu_z - EI\omega_x\omega_z + GJ\omega_x\omega_z \\ 0 \\ GA\nu_y\omega_z - EA\omega_y(\nu_z - 1) \\ EA\omega_x(\nu_z - 1) - GA\nu_x\omega_z \\ GA(\nu_x\omega_y - \nu_y\omega_x) \end{bmatrix} \tag{2.79}$$

$$W_{ext} = \begin{bmatrix} EI\omega_x \\ EI\omega_y \\ GJ\omega_z \\ GA\nu_x \\ GA\nu_y \\ EA(\nu_z - 1) \end{bmatrix} \tag{2.80}$$

where the components of ξ been used, $\xi = [\omega_x, \omega_y, \omega_z, \nu_x, \nu_y, \nu_z]^T$.

Some interesting cases arise when we look for ξ where $\bar{W} = 0$. There appear to be two cases: pure bending and the combination of torsion and extension. These constant deformations can be achieved with only point loads at the tip. For bending, as long as only a moment about the x or y-axis is applied it will be pure bending. For torsion and extension, only a torsional moment and an axial force may be applied. All other situations need some sort of constant distributed wrench to satisfy the constant ξ assumption.

2.6 Numerical Implementations

To solve the statics we need to integrate the system of ODEs and we want them to be as accurate as possible. One consideration to be careful of is the integration of the rotation matrix, R. Rotation matrices have the property $det(R) = 1$; however, if we were to integrate the ODE $R' = R\hat{\omega}$ with a standard Runge-Kutta (RK) integrator the determinant would drift and cause the rotation to not be proper and gradually become more inaccurate [59]. In order to preserve the properties of R we

need to use an integrator designed to preserve them. One class of methods are Lie group integrators that are designed to preserve the structure of the group elements. As with all numerical methods we only preserve the structure up to numerical precision.

2.6.1 Lie Group Integrators

Lie groups generally have some structural restrictions placed on them, such as $SO(3) = \{R|R^T R = RR^T = I, det(R) = 1\}$ which integrators will only maintain when designed to. With the rotation example we have $R' = R\hat{\omega}$ and if this were integrated with the explicit Euler scheme we get:

$$R_{i+1} = R_i(I + \Delta s\hat{\omega}) \tag{2.81}$$

which shows that $det(R_i) \neq 1$ making the rotation no longer proper. This feature is present in all RK integrators.

For integration schemes that preserve the Lie group structure we use Lie group integrators. There have been several approaches to tackling the structure preserving problem, but the basic idea is to use the group action (for matrix Lie groups this is just multiplication) to preserve the structure between steps and to generate an element of the group for the stepping [60]. So, it will look something like:

$$R_{i+1} = R_i A_i \tag{2.82}$$

where $A_i \in SO(3)$ is constructed to be a group element. One way to guarantee that A_i belongs to the group is to use local diffeomorphisms and the Lie algebra. Since the algebra elements make up a vector space the elements will always remain a member of the algebra with the arithmetic used and the diffeomorphisms guarantee the algebra is mapped to a group. So, working with the algebra and then mapping to the group structure when needing to step the integrator will always preserve the group structure.

One approach to integrating Lie groups are the Runge-Kutta-Munthe-Kaas (RKMK) integrators [60, 61]. RKMK integrators are an extension of RK methods that preserve the Lie group

structure. The general idea is as stated above, do the integration on the algebra and transform the algebra elements to the group elements. For the rod system this begins with approximating g as:

$$g = g_i \exp(\theta), \theta(0) = 0 \tag{2.83}$$

which states that g local to g_i can be approximated by the diffeomorphism, $\exp$, with the algebra element, θ, making sure that at $\theta(0)$ we have $g = g_i$. Taking the derivative and using (2.6) we get:

$$\theta' = \mathrm{dexp}^{-1}_{-\theta} \xi \tag{2.84}$$

as we did before in the constant ξ example in the statics discussion.

Since $\theta \in se(3)$ it can be integrated with an RK scheme just fine and the result can be used to integrate g. Often dexp will either not have an analytic form or is too expensive to compute, so an approximate form will be used. Given that the underlying RK scheme already has a given accuracy we want the approximation of dexp to not influence it. For an RK scheme of order p then the dexp approximation order, q, must be $q \geq p - 2$ to maintain the order of the RK scheme [59]. For example, this means for 2nd-order RK schemes we can use $\mathrm{dexp} \approx I$, which is quite useful as it can save some computation time.

The RKMK scheme thus preserves the group structure by using group multiplication and the diffeomorphism to map algebra elements to group elements. Any integration can then be done on the algebra and still maintain the structure.

For the static rod system the process is to integrate ξ' and θ' using the same RK scheme and using the exponential map and group multiplication to preserve g. As an example if the implicit midpoint scheme is used ($y_{i+1} = y_i + \Delta t f(t_i + \Delta t/2, (y_{i+1} + y_i)/2)$) the resulting scheme is:

$$\xi_{i+1} = \xi_i + \Delta s \xi'(s_i + \frac{\Delta s}{2}, \tilde{\xi}, \tilde{g}) \tag{2.85}$$

$$\theta = \Delta s \theta'(\tilde{\theta}, \tilde{\xi}) \tag{2.86}$$

$$g_{i+1} = g_i \exp(\theta) \tag{2.87}$$

$$\tilde{\xi} = \xi_i + \frac{\Delta s}{2} \xi'(s_i + \frac{\Delta s}{2}, \tilde{\xi}, \tilde{g}) \tag{2.88}$$

$$\tilde{\theta} = \frac{\Delta s}{2} \theta'(\tilde{\theta}, \tilde{\xi}) \tag{2.89}$$

$$\tilde{g} = g_i \exp(\tilde{\theta}) \tag{2.90}$$

where ˜ stands for intermediate values in the integration process.

Using the approximation $\mathrm{dexp} \approx I$, since this is a 2nd-order scheme, we get:

$$\xi_{i+1} = \xi_i + \Delta s \xi'(s_i + \frac{\Delta s}{2}, \frac{\Delta s}{2}(\xi_{i+1} + \xi_i), g_i \exp(\frac{\Delta s}{4}(\xi_{i+1} + \xi_i))) \tag{2.91}$$

$$g_{i+1} = g_i \exp(\frac{\Delta s}{2}(\xi_{i+1} + \xi_i)) \tag{2.92}$$

All RK schemes can be turned into an RKMK scheme [61] and in the tests we will show the results of several schemes.

2.6.2 Pseudospectral Methods

In numerical integration there is always the concern of how close the solution is to the correct answer and what is the discretization necessary for getting a desired accuracy. One method that typically gives the best or near best accuracy per number of sample points is the pseudospectral (PS) method [62]. PS methods are very similar to Galerkin or spectral methods except the resulting integrals are evaluated using Gaussian quadrature rather than analytically. The idea for PS methods are to pose the problem in weak form and assume that both the solution and variation, or test function, are a family of basis functions. Then to find the coefficients associated with the basis functions the resulting integrals are evaluated by Gaussian quadrature, which is determined from

the choice of basis functions. The benefit to using PS methods are that high accuracy is achieved with a low number of discrete points [62].

Another view of PS methods is to consider the PS method as a finite difference method with only one segment and a high order interpolating polynomial rather than several segments with low order polynomials. This interpretation allows the solution process to be posed like a finite difference method where the difference matrix is dense rather than sparse.

The basic idea behind a Galerkin method [62] is to take an ODE, $\dot{y} = f(y,t)$, and write it in weak form:

$$\langle u, \dot{y} \rangle = \langle u, f \rangle \tag{2.93}$$

where u is some function that satisfies the boundary conditions imposed on y and the inner product here is on functions, $\langle a(x), b(x) \rangle = \int a(x)b(x)dx$. Using a basis of orthonormal functions, ϕ, approximate $u = \sum \phi_i$ and $y = \sum \alpha_i \phi_i$ where α_i are the coefficients for y's approximation. If the basis functions are well selected these integrals can be evaluated analytically and the coefficients, α_i, can be solved for with high accuracy. This is the Galerkin method; however, most problems will not have nice integrals and will need to be numerically integrated.

A common and highly accurate technique for numerical integration is Gaussian quadrature which approximates an integral as:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=0}^{N} w_i f(x_i) \tag{2.94}$$

where w_i are weighting coefficients, x_i are the sample points, and N is the number of points used. Then, to determine the weights and sample points find the w_i and x_i that exactly integrate polynomials up to order $2N+1$. It is important to note that Gaussian quadrature is defined on the domain $[-1, 1]$ and anytime it is used the domain of the function needs to be adjusted to this range.

Combining these techniques results in the pseudospectral method which has nearly the accuracy of a Galerkin method [62], but is more computationally feasible. For problems where the domain is finite and the boundary conditions are not periodic, such as the rod, the best choice

for basis functions are Chebyshev polynomials of the first kind, $T_n(x)$ [62]. The sample points are then $x_i = cos(\frac{i\pi}{N}), i = 0, \ldots, N$ the Chebyshev-Lobatto grid. Another choice for the sample points is viable; however, this grid is more convenient to use given the boundary conditions. The Chebyshev polynomials are defined as:

$$T_n(cos(x)) = cos(nx) \tag{2.95}$$

or an equivalent recursive definition:

$$T_0(x) = 1 \tag{2.96}$$

$$T_1(x) = x \tag{2.97}$$

$$T_i(x) = 2xT_{i-1}(x) - T_{i-2}(x) \tag{2.98}$$

While it is possible to directly expand the functions in terms of T_n it is a bit more convenient to use the equivalent cardinal basis [62] in the finite difference interpretation:

$$y(x) = \sum_{i=0}^{N} y_i C_i(x) \tag{2.99}$$

$$C_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{2.100}$$

where C_i are the cardinal basis functions and the x_j are the Chebyshev nodes mentioned before.

Using these approximations we can write the ODE as:

$$Dy = f(y, t) \tag{2.101}$$

where D is the differentiation matrix for the cardinal basis on the Chebyshev nodes and all the function, $f(y, t)$, evaluations occur at the nodes. The differentiation matrix is:

$$D = \begin{cases} \frac{1+2N^2}{6} & i = j = 0 \\ -\frac{1+2N^2}{6} & i = j = N \\ -\frac{x_j}{2(1-x_j^2)} & i = j, 0 < i, j < N \\ (-1)^{i+j} \frac{p_i}{p_j(x_i - x_j)} & i \neq j \end{cases} \tag{2.102}$$

where $p_0 = p_N = 2$, $p_j = 1$ otherwise [62].

To incorporate boundary conditions into the system we replace the rows of D and f corresponding to the endpoints and set them to be the boundary conditions (e.g., making the tip load balance). This replacement happens for the boundary conditions on ξ and θ individually.

For the static rod equations this amounts to:

$$D\xi = \frac{L}{2} K^{-1} (ad_\xi^T K \Delta\xi - \bar{W}) \tag{2.103}$$

$$D\theta = \mathrm{dexp}_{-\theta}^{-1} \xi \tag{2.104}$$

$$g_i = g_0 \exp(\theta_i) \tag{2.105}$$

where the D matrices are modified to incorporate the boundary conditions and $\frac{L}{2}$ is a scaling factor from changing the domain of s from $[0, L]$ to $[-1, 1]$. It is important to note that these are nonlinear and thus must be iteratively solved for ξ and θ. To make sure there is no degradation in the solution the analytic form of dexp is used (2.72).

2.7 Results

In the implementation of the statics we want to compare the performance between RKMK schemes and PS. Since all RK schemes can be converted to RKMK schemes we will compare several versions of RKMK as well with different orders of accuracy and solving approaches. We expect that the pseudospectral method is the most accurate for the number of points, the higher order RKMK schemes are more accurate than the low order schemes, and the explicit schemes are

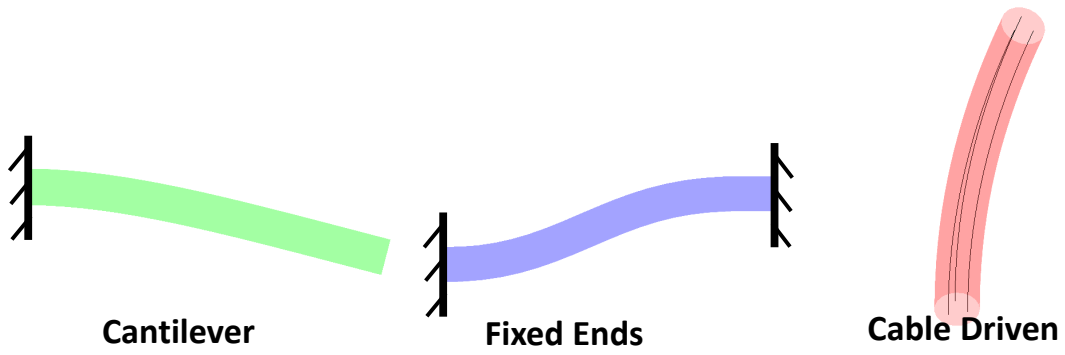


Figure 2.3: The cases considered for the static manipulator. We have the cantilever case, the fixed ends case, and the cable-driven case.

the fastest to compute. To compare the choices we use a cantilevered rod subject to gravity and a rod with both ends fixed as test problems. We also report the cable-driven static case for the sake of establishing the workspace.

For the RKMK schemes we will compare the explicit Euler, implicit Euler, explicit midpoint, and implicit midpoint. The Euler methods are of order 1 and the midpoints are 2nd-order. The discretization schemes are:

$$\text{explicit Euler: } y_{i+1} = y_i + \Delta t f(t_i, y_i) \tag{2.106}$$

$$\text{implicit Euler: } y_{i+1} = y_i + \Delta t f(t_i + \Delta t, y_{i+1}) \tag{2.107}$$

$$\text{explicit midpoint: } y_{i+1} = y_i + \Delta t f(t_i + \Delta t/2, y_i + \Delta t/2 f(t_i, y_i)) \tag{2.108}$$

$$\text{implicit midpoint: } y_{i+1} = y_i + \Delta t f(t_i + \Delta t/2, \frac{y_i + y_{i+1}}{2}) \tag{2.109}$$

First we will review the physics for the cantilevered rod and the rod with fixed ends and then give the numerical results. Throughout we assume a cylindrical rod that is initially straight ($\xi = [0, 0, 0, 0, 0, 1]^T$) with the properties found in Table 2.1.

2.7.1 Cantilevered Rod

A cantilevered rod is a rod that has been fixed to a wall at one end and the other end left to hang freely due to gravity. Here the rod is fixed to a wall such that it is perpendicular to the direction of gravity.

The physics for a cantilevered rod can be stated as:

$$0 = \bar{W}_{grav} - ad_{\xi}^{T} K\Delta\xi + (K\Delta\xi)' \tag{2.110}$$

$$g(0) = I \tag{2.111}$$

$$\xi(L) = \xi^{*} \tag{2.112}$$

We will make the z-axis horizontal and have gravity be pointing in the x-direction, $g_r = [-9.81, 0, 0]^T$.

For the cantilevered rod the boundary conditions are that the base is fixed, $g(0) = I$, and that the tip is free, $\xi(L) = \xi^{*}$. Due to the mixed boundary conditions the RKMK schemes need to use a shooting method to solve the problem. The procedure begins by guessing the value for ξ at the base, integrating over s using the RKMK scheme, checking if $\xi(L) = \xi^{*}$, and repeating until convergence. For the PS method the boundary conditions simply have to be added to D and F and the system solved.

2.7.2 Fixed Ends

For the rod with both ends fixed we will consider a system with no external forces for simplicity and the tip configuration being a translation from the neutral position. The system is thus:

$$0 = (K\Delta\xi)' - ad_\xi^T K\Delta\xi \tag{2.113}$$

$$g(0) = I \tag{2.114}$$

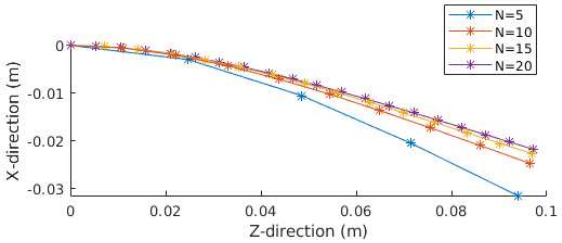
$$g(L) = \begin{bmatrix} I_{3\times3} & p_L \\ 0 & 1 \end{bmatrix}, p_L = \begin{bmatrix} 0.1L \\ 0 \\ L \end{bmatrix} \tag{2.115}$$

With both ends fixed the RKMK integrator still needs the shooting method to solve, but we need to define the error in the tip configuration error. The error is made of both the position error and the orientation error, the position error is $p_{err} = p(L) - p_{des}$ and the orientation error can be defined as $\phi_{err} = \log(R_{des}^T R(L))\check{}$. For the PS method the normal computations are used, but one row of the equations is replaced by the configuration error.

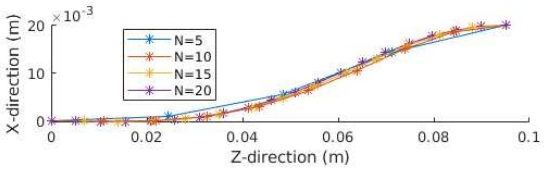
Results

In testing the static solvers of the RKMK integrators and the PS scheme we want to see how many points are necessary for convergence and if the solutions are the same. Testing the solutions for the cantilevered rod and the fixed ends rod are found in Figure 2.4. The midpoint methods converged faster than the Euler methods on all tests as they are second order methods compared first order, as expected. The PS method converged very quickly in the cantilever case, but not quite as quickly in the fixed ends case. Only one solution failed to solve which was the fixed ends implicit midpoint scheme with $N = 15$, which could be fixed by an improved initial guess for $\xi(0)$. It is also notable that the determinants of the rotations are preserved as expected, the largest deviation measured for all schemes was about 5×10^{-16} which is within numerical precision, so all schemes can be considered to preserve the group structure.

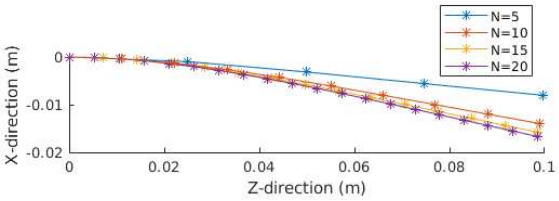
As for computation speed, although it isn't too important for the statics, the explicit schemes are a bit faster than PS schemes and the implicit RKMK schemes are the slowest. Deciding between using an explicit RKMK scheme and the PS scheme is a toss up and mostly preference. However, looking ahead to the dynamics simulations the RKMK integrators are more amenable to dynamics



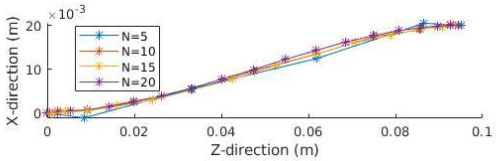
(a) Cantilever, explicit Euler.



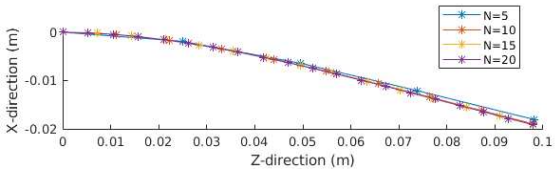
(b) Fixed ends, explicit Euler.



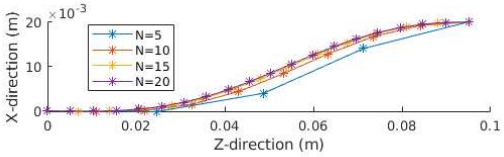
(c) Cantilever, implicit Euler.



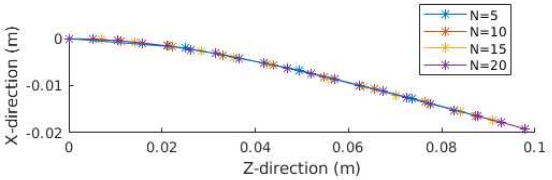
(d) Fixed ends, implicit Euler.



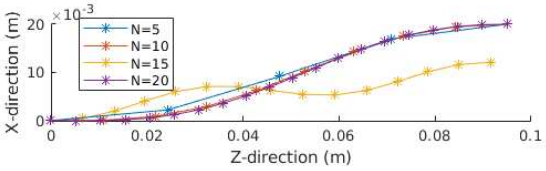
(e) Cantilever, explicit midpoint.



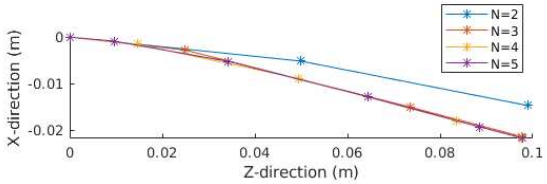
(f) Fixed ends, explicit midpoint.



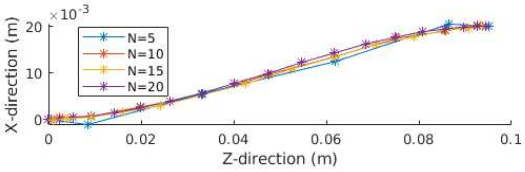
(g) Cantilever, implicit midpoint.



(h) Fixed ends, implicit midpoint.



(i) Cantilever, pseudospectral.



(j) Fixed ends, pseudospectral.

Figure 2.4: The solutions and convergence of the various different schemes for the cantilever hanging under gravity and the beam with both ends fixed.

formulations while the PS method has some complications with the implementation and derivation of a dynamic numerical scheme and loses its appeal for use in the dynamics.

2.7.3 Static Workspace

Traditionally robots have what is known as a workspace, the region in which they can move their end-effector, which is useful for characterizing the behavior and limitations of the robot. Workspaces don't always have easy to compute forms, but a decent representation can usually be obtained using something like a convex hull. However, the equivalent workspace for soft robots has no analytic form and is not convex making it more difficult to quantify. Traditional robots have their workspace determined by the limits of their geometry; however, soft robots do not have the same geometric restrictions and without limits on their actuators can have theoretically unbounded workspaces ignoring material limits. We need some definition of the workspace for the sake of coming up with feasible control tasks. To quantify the workspace we need actuator limits. For our testing we use cables and assume that the tensions cannot exceed 0.5 N to develop an approximate analytic form of the workspace.

To define the workspace we could simply sample the workspace and find a surface that bounds the sampled points. However, the workspace will not be convex making it harder to define the surface and not having an analytic form for the workspace makes it harder to determine and generate points inside and outside the workspace. To establish a workspace we pick a simplified model of the manipulator that has an analytic tip position and sample the workspace to solve the parameters that bound the workspace.

For the simplified model we assume the primary deformations are bending and extension then we model the workspace using a rod with constant bending and constant extension with no other deformations. This gives:

$$R = R_x(\omega s) \tag{2.116}$$

$$\xi = \begin{bmatrix} 0 \\ \omega \\ 0 \\ 0 \\ 0 \\ \epsilon + 1 \end{bmatrix} \tag{2.117}$$

where ω is the bending deformation, ϵ is the stretch, and R_x is a rotation about the x-axis. Then if we assume that there is rotational symmetry about the z-axis we get:

$$R = R_z(\phi)R_x(\omega s) \tag{2.118}$$

$$p' = R \begin{bmatrix} 0 \\ 0 \\ \epsilon + 1 \end{bmatrix} \tag{2.119}$$

where ϕ is the angle about the z-axis. The tip position can be solved for:

$$p_{tip} = \begin{bmatrix} 2sin(\frac{\omega L}{2})^2 sin(\phi) \\ -2sin(\frac{\omega L}{2})^2 cos(\phi) \\ sin(\omega L) \end{bmatrix} \frac{1+\epsilon}{\omega} \tag{2.120}$$

thus the workspace is established when the bounds for ω and ϵ are determined. We know that $\phi \in [0, 2\pi]$ and due to symmetry $\omega \in [-\omega_{bound}, \omega_{bound}]$. For the stretch it will typically be asymmetric because either the actuators compress the body or extend it, $\epsilon \in [\epsilon_{low}, \epsilon_{high}]$. It is important to note that there is an assumed symmetry in the placement of the actuators that should hold true in our testing, but will not hold for a general manipulator. To establish a rough workspace we need to determine ω_{bound}, ϵ_{low}, and ϵ_{high} so that the workspace encapsulates all the statically reachable tip positions. This can be done through sampling the workspace and finding the necessary boundaries.

Then the workspace coordinates (ω, ϕ, ϵ) need to be found in terms of the Cartesian positions.

$$\phi = atan(-\frac{p_x}{p_y}) \tag{2.121}$$

$$\omega = \frac{2}{L} atan(\frac{\sqrt{p_x^2 + p_y^2}}{p_z}) \tag{2.122}$$

$$\epsilon = \frac{p_z \omega}{sin(\omega L)} - 1 \tag{2.123}$$

where p_x, p_y, and p_z are the components of the tip position.

Both the workspace and the workspace coordinates have an issue when $\omega = 0$, in the limit these expressions become:

$$p_{tip} = \begin{bmatrix} 0 \\ 0 \\ L(1+\epsilon) \end{bmatrix} \tag{2.124}$$

$$\omega = 0 \tag{2.125}$$

$$\phi = 0 \tag{2.126}$$

$$\epsilon = p_z/L - 1 \tag{2.127}$$

To sample the workspace we solve the statics for several inputs covering the range of valid input tensions and expand the bounds of the workspace to encompass all the sampled points, the resulting workspace can be seen in Figure 2.5. This formulation will always give an umbrella shaped volume for the workspace akin to a warped and relatively flat cylinder.

It is worth noting that this overestimates the workspace, but it doesn't seem to cause any significant problems in our use cases of sampling points and testing the control.

2.8 Summary

In this chapter we developed the kinematics and statics for the Cosserat rod, discussed different actuator loadings, and demonstrated the numerical implementations for the system. The kinematics

Table 2.1: The relevant assumed material properties and geometry of the soft rod used in the static solver tests.

$E(MPA)$	$G(MPa)$	$\rho(kg/m^3)$	$D(cm)$	$L(cm)$
1	0.33	1000	1	10

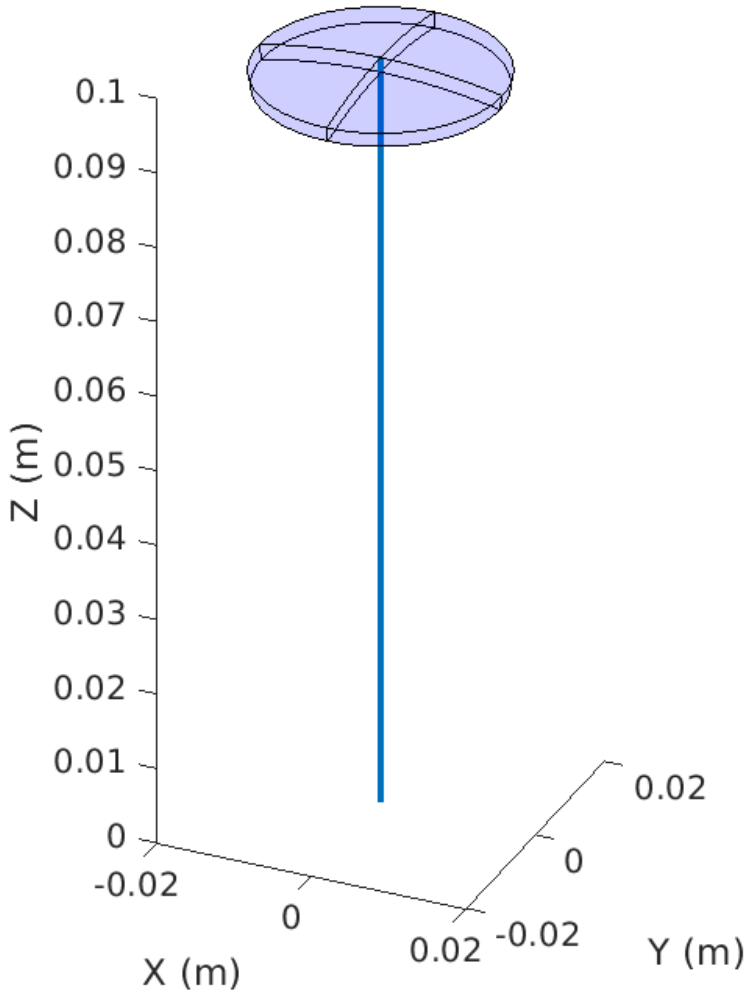


Figure 2.5: The workspace for the static manipulator driven by cables and limited to a tension of 0.5 N.

for the system are described by the centerline s, the directors, and the configurations using the rigid body displacements of $SE(3)$. The statics are determined using the principle of virtual work to balance the internal strain and the external loading. The most common types of actuators, cables and pneumatics, were discussed and the forms of the loading derived. We also discussed the usage of artificial muscles and some potential difficulties with the numerical implementation due to different types of coupling. Finally, we discussed how to solve the static rod system as it is a boundary value problem and how to preserve the Lie group structure, primarily rotations R, during the integration process.

Chapter 3

Soft Manipulator Dynamics

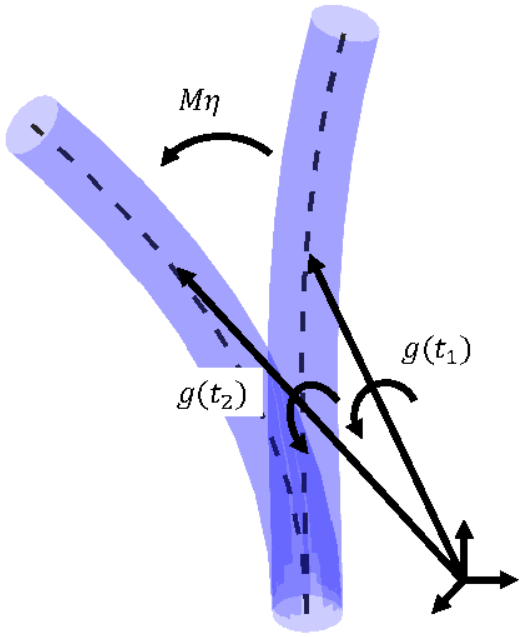


Figure 3.1: A diagram representing the change of the configuration between time t_1 and t_2 due to momentum, $M\eta$.

The dynamics for the Cosserat rod can be derived from Hamilton's principle (HP) [53]. HP states that the path taken by a dynamic system extremizes the action of the system or in other words the path taken is the one that leaves the variation of the action stationary. This means that the dynamics of the system can be derived using VC. Technically this applies only to conservative systems, but it can be modified to nonconservative systems using the principle of Virtual Work and is referred to as the Lagrange-D'Alembert principle [63]. Using VC, the Lagrange-D'Alembert principle (which will be referred to as Hamilton's principle) can be stated as:

$$\delta(\int_0^\tau \mathcal{L}dt) + \int_0^\tau \langle \delta h, W \rangle dt = 0 \tag{3.1}$$

where $\mathcal{L}$ is the Lagrangian of the system and τ is the time horizon of the problem. We consider the Lagrangian to be $L = T - U$ where T is the kinetic energy and U is the strain energy (2.28) as defined in the statics. We also need to recall that the boundary conditions in time cannot be perturbed so $\delta h(0) = \delta h(\tau) = 0$.

3.1 Kinetic Energy

For the kinetic energy, T, we have the total kinetic energy of every point in the body which can be viewed as the integral of the kinetic energy per unit volume over the entire volume of the body.

$$T = \int \frac{1}{2}\langle \dot{p}_G, \rho \dot{p}_G \rangle dV \tag{3.2}$$

where V is the volume of the body and ρ is the density of the body. Rearranging the equations, assuming the density is constant, and using the fact that κ and ψ are only dependent on s (not x or y) we get:

$$T = \frac{1}{2}\rho \int (A\langle \kappa, \kappa \rangle - \langle \psi, (\int \hat{r}^2 dA)\psi \rangle - 2\langle v, (\int \hat{r} dA) w \rangle) ds \tag{3.3}$$

Then since the centerline is defined to pass through the centroid of the cross sections terms like x or xy will integrate to 0 due to symmetry (2.5) and the kinetic energy simplifies to:

$$T = \frac{1}{2}\rho \int (A\langle \kappa, \kappa \rangle + \langle \psi, (\int \begin{bmatrix} x^2 & 0 & 0 \\ 0 & y^2 & 0 \\ 0 & 0 & x^2 + y^2 \end{bmatrix} dA)\psi \rangle) ds \tag{3.4}$$

which can be rewritten in matrix form as:

$$T = \frac{1}{2} \int \langle \eta, M\eta \rangle ds \tag{3.5}$$

where $M = \rho * \text{diag}(I_x, I_y, J, A, A, A)$ is the inertia matrix for a cross section.

To evaluate the variation we use essentially the same process as the derivation for the statics. The variation of η derives from the variation of the configuration and the definition of the temporal twist (1.32).

$$\delta\eta = \dot{\delta h} + ad_\eta \delta h \tag{3.6}$$

Using $\delta\eta$, HP (3.1), the previously defined variation of the strain energy (2.30), and work (2.18) the weak form of the dynamics becomes:

$$0 = \int_0^\tau (\langle \delta h, K\Delta\xi \rangle|_0^L + \int_0^L \langle \delta h, ad_\eta^T M\eta - \dot{M}\eta - ad_\xi^T K\Delta\xi + (K\Delta\xi)' + \bar{W}\rangle ds)dt \tag{3.7}$$

where integration by parts has been used as well as $\delta h(0) = \delta h(\tau) = 0$ to simplify the expression. The strong form is:

$$ad_\eta^T M\eta - \dot{M}\eta - ad_\xi^T K\Delta\xi + (K\Delta\xi)' + \bar{W} = 0 \tag{3.8}$$

The full description of the dynamics is then made up of (2.15), (2.16), and either (3.7) or (3.8) with the associated boundary and initial conditions.

One thing we note from (3.8) is that the transpose of the adjoint is the coadjoint map which implies that $M\eta$ and $K\Delta\xi$ are the conjugate momenta to η and ξ respectively. The conjugate momenta can be interpreted as the momentum, $\mu = M\eta$, and the stress, $\lambda = K\Delta\xi$, and we can write the dynamics as:

$$(ad_\eta^T - \frac{d}{dt})\mu - (ad_\xi^T - \frac{d}{ds})\lambda + \bar{W} = 0 \tag{3.9}$$

This could be a useful observation for stating the dynamics as a Hamiltonian system; however, we don't make use of it.

3.2 External Loading

Like in the statics all loads that do not derive from the kinetic or strain energy are included as external load components of $\bar{W}$. In the dynamics we have more effects such as viscosity and drag or dynamic behaviors of the actuators. Here a few common loads are presented.

3.2.1 Viscosity

The viscosity effects are dependent on the the time rate change of the strain in the body and there exists many different approaches to modeling the effect. Here we will use the equivalent of the Kelvin-Voigt model following the derivation in [64]. The derivation is very similar to the stiffness derivation where we assume a linear model of the viscosity and integrate over the body. The elastic term is already included in the model as $K\Delta\xi$, so the viscosity term is:

$$\bar{W}_{vis} = V\dot{\xi} \tag{3.10}$$

where $V = \alpha * \text{diag}([3I_x, 3I_y, J, A, A, 3A])$ with α as the shear viscosity of the material and using Trouton's ratio to say that the extensional viscosity is three times the shear viscosity.

3.2.2 Drag

For drag effects there are typically two regimes to consider: laminar and turbulent. For laminar drag the force is proportional to the speed and applies when the Reynolds number is small and turbulent drag is proportional to the square of the speed when the Reynolds number is high. The drag coefficients usually have to be found empirically, but the forms of the drag can be represented as:

$$\bar{W}_{drag,lam} = -\rho_{fluid} D_{lam} \eta \tag{3.11}$$

$$\bar{W}_{drag,turb} = -\rho_{fluid} D_{turb} \|v\| \eta \tag{3.12}$$

where D are the drag matrix. Typically no moment will be produced by the drag which should be reflected in the drag matrices.

3.2.3 Embedded Rod

For the behavior of an actuator embedded in the body a fairly general approach is to consider the actuator as a rod that generates its own internal wrench. This allows for the inclusion of the actuator's stiffness and inertia naturally into the system and would be the process for extending the actuator formulations in the statics to the dynamics.

For the kinematics of the embedded rod the configuration will typically just be a translation from the main rod's centerline.

$$g_{act} = gT_{act} = g \begin{bmatrix} I_{3\times3} & r_{act} \\ 0 & 1 \end{bmatrix} \tag{3.13}$$

where r_{act} is the displacement from the centerline. The twists follow as:

$$g'_{act} = g_{act}\hat{\xi}_{act} \tag{3.14}$$

$$\dot{g}_{act} = g_{act}\hat{\eta}_{act} \tag{3.15}$$

The dynamics of the actuator are then the same form as the body dynamics.

$$ad^T_{\eta_{act}} M_{act}\eta_{act} - M_{act}\dot{\eta}_{act} - ad^T_{\xi_{act}} K_{act}\Delta\xi_{act} + (K_{act}\Delta\xi_{act})' + \bar{W}_{gen} + \bar{W}_{body} = 0 \tag{3.16}$$

where $\bar{W}_{gen}$ is the distributed wrench generated by the actuator and $\bar{W}_{body}$ will be the distributed wrench due to the main body. The distributed wrench due to the actuator in the body, $\bar{W}_{act}$, can then be written as:

$$\bar{W}_{act} = Ad^T_{T^{-1}_{act}} \bar{W}_{body} \tag{3.17}$$

which allows the actuator dynamics to be incorporated into the dynamics of the body.

To relate the actuator twists and body twists we can derive them as:

$$g'_{act} = g_{act}\hat{\xi}_{act} = gT_{act}\hat{\xi}_{act} = g(\hat{\xi}T_{act} + T_{act}\hat{\xi}_T) \tag{3.18}$$

$$\xi_{act} = Ad_{T_{act}^{-1}}\xi + \xi_T \tag{3.19}$$

where ξ_T is the spatial twist associated with T_{act}. The temporal twist follows similarly except we know that $\dot{T}_{act} = 0$ since the location in the body should not be changing.

$$\eta_{act} = Ad_{T_{act}^{-1}}\eta \tag{3.20}$$

3.3 Numerical Simulations

Just like in the statics we do not have analytic solutions to the behavior and thus need numerical simulations. For the numerical simulations we also have the concern of preserving the Lie group structure and have the new objective of accurately mimicking the physics of the system. Primarily what this means is that we do not want to introduce artificial damping to the simulations. Typically numerical integrators that are not designed for preserving energy will introduce some amount of numerical damping leading to the energy either dissipating from the system or growing to infinity even when the physics is conservative [59]. Now, our system will have dissipation from viscosity and other nonconservative loads, but we do not want to introduce numerical damping so that the simulations can match the described behavior as best as possible. This motivates us to use special types of integrators known as symplectic integrators [59].

Symplectic integrators preserve the symplectic structure of the system, which means that the Hamiltonian or total energy is preserved. There are various techniques for creating these types of integrators, but one convenient approach is variational integrators [59].

In this section we will develop several integrators. The first will be an explicit variational integrator for the Cosserat rod. Next, we will impose the Kirchhoff restriction to reduce numerical stiffness. Finally, we will develop an implicit scheme similar to [40].

3.3.1 Explicit Variational Integrator

The appeal of variational integrators is the fact that they are naturally symplectic [59]. Variational integrators typically start with Hamilton's principle (or the Lagrange-D'Alembert principle when nonconservative), discretize the integrals, and take the variation of the discrete equations to extremize the system. This process then defines the integration scheme.

An example derivation starts with the Lagrange-D'Alembert principle (3.1):

$$\delta(\int_0^T \mathcal{L}(q,\dot{q},t)dt) + \int_0^T \langle \delta q, W\rangle dt = 0 \tag{3.21}$$

where $\mathcal{L}$ is the Lagrangian, q are the generalize coordinates, $\dot{q}$ the generalized velocities, δq the variation of q, and W the external loads. Next, over a short time span, Δt, approximate the integrals:

$$\mathcal{L}_D(q_i, q_{i+1}) \approx \int_{t_i}^{t_i+\Delta t} \mathcal{L}(q,\dot{q},t)dt \tag{3.22}$$

$$\langle \delta q_i, W_i^-(q_i, q_{i+1})\rangle + \langle \delta q_{i+1}, W^+(q_i, q_{i+1})\rangle \approx \int_{t_i}^{t_i+\Delta t} \langle \delta q, W\rangle dt \tag{3.23}$$

where $\mathcal{L}_D$ is the approximation of the action, and W^- and W^+ are chosen such that they approximate the virtual work. Taking the variation, $\mathcal{L}_D$ becomes:

$$\delta(\mathcal{L}_D) = \frac{\partial \mathcal{L}_D}{\partial q_i}\delta q_i + \frac{\partial \mathcal{L}_D}{\partial q_{i+1}}\delta q_{i+1} \tag{3.24}$$

Using the fact that the summation of the variation of these approximated integrals, $\sum \mathcal{L}_D$, should be 0 and rearranging to collect terms dependent on δq_i the resulting equation is:

$$0 = \frac{\partial \mathcal{L}_D(q_i, q_{i+1})}{\partial q_i} + \frac{\partial \mathcal{L}_D(q_{i-1}, q_i)}{\partial q_i} + W^-(q_i, q_{i+1}) + W^+(q_{i-1}, q_i) \tag{3.25}$$

which gives an implicit definition for the time integration scheme, $q_{i+1} = f(q_{i-1}, q_i)$.

For formulating a variational integrator for the Cosserat rod we want to preserve the Lie group structure. For this we use an RKMK integrator for the kinematics and derive the variational integrator from there. Here we will develop a low order integrator in space and time following similarly to [63].

For the kinematics we have stepping in space and stepping in time, for both we will be using explicit Euler RKMK integrators.

$$g_{i+1}^{j} = g_i^j \exp(\Delta t \hat{\eta}_i^j) \tag{3.26}$$

$$g_i^{j+1} = g_i^j \exp(\Delta s \hat{\xi}_i^j) \tag{3.27}$$

where the subscripts denote the point in time and the superscripts denote the point in space. Next, we want to discretize the Lagrange-D'Alembert principle (3.1) for the rod. For this discretization over space and time we will discretize over a square of space-time due to the double integration, Figure 3.2. To approximate the integrals we will use the lower left corner values η_i^j and ξ_i^j.

$$\mathcal{L}_D(\eta_i^j, \xi_i^j) = \Delta t \Delta s (\frac{1}{2}\langle \eta_i^j, M\eta_i^j \rangle - \frac{1}{2}\langle \Delta\xi_i^j, K\Delta\xi_i^j \rangle) \tag{3.28}$$

where a Riemann integration from the lower left corner has been used to approximate the double integral.

As a simple approximation to the virtual work term we can use:

$$\int_{t_i}^{t_i+\Delta t}\int_{s_j}^{s_j+\Delta s} \langle \delta h, \bar{W} \rangle ds dt \approx \Delta t \Delta s \langle \delta h_{i+1}^{j+1}, \bar{W}_i^j \rangle \tag{3.29}$$

this is a convenient form as the resulting scheme will be explicit and simple to compute allowing for a fairly efficient implementation.

Evaluating the variation of the discrete Lagrange-D'Alembert principle comes to:

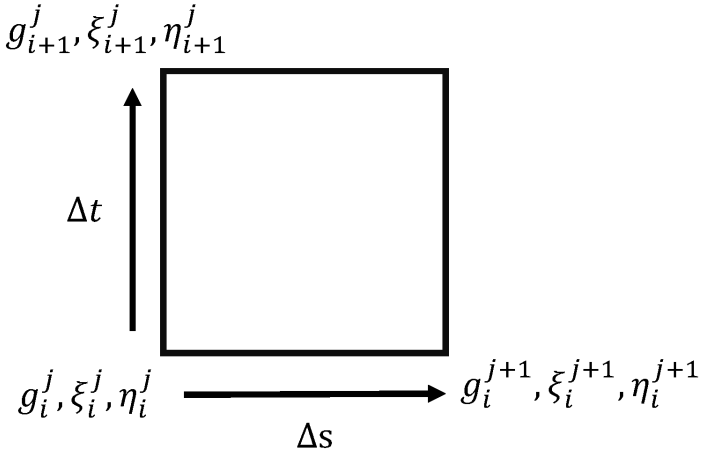


Figure 3.2: This represents the discretization of space and time in the variational integration scheme.

$$0 = \Delta t \Delta s \sum_i \sum_j (\langle \delta \eta_i^j, M \eta_i^j \rangle - \langle \delta \xi_i^j, K \Delta \xi_i^j \rangle + \langle \delta h_{i+1}^{j+1}, \bar{W}_i^j \rangle) \tag{3.30}$$

To determine the variations of η and ξ we can use the variations of the kinematics (3.26) and (3.27). First we define the variation of the discrete configuration:

$$\delta g_i^j = g_i^j \hat{\delta h_i^j} \tag{3.31}$$

then taking the variations of (3.26) and (3.27) we get [63]:

$$g_{i+1}^j \hat{\delta h_{i+1}^j} = g_i^j \hat{\delta h_i^j} \exp(\Delta t \hat{\eta_i^j}) + g_i^j \operatorname{dexp}_{\Delta t \eta_i^j} \Delta t \hat{\delta \eta_i^j} \exp(\Delta t \hat{\eta_i^j}) \tag{3.32}$$

$$g_i^{j+1} \hat{\delta h_i^{j+1}} = g_i^j \hat{\delta h_i^j} \exp(\Delta s \hat{\xi_i^j}) + g_i^j \operatorname{dexp}_{\Delta s \xi_i^j} \Delta s \hat{\delta \xi_i^j} \exp(\Delta s \hat{\xi_i^j}) \tag{3.33}$$

which can be rearranged to:

$$\delta\eta_i^j = \frac{1}{\Delta t}\operatorname{dexp}_{\Delta t\eta_i^j}(-\delta h_i^j + Ad_{\exp(\Delta t\eta_i^j)}\delta h_{i+1}^j) \tag{3.34}$$

$$\delta\xi_i^j = \frac{1}{\Delta s}\operatorname{dexp}_{\Delta s\xi_i^j}(-\delta h_i^j + Ad_{\exp(\Delta s\xi_i^j)}\delta h_i^{j+1}) \tag{3.35}$$

The variations are substituted into (3.30). Reordering terms and using the fact that δh_i^j must be an arbitrary value, we get that the variational dicretization scheme for the rod is:

$$0 = \frac{1}{\Delta t}(-\mu_i^j + Ad^T_{\exp(\Delta t\eta_{i-1}^j)}\mu_{i-1}^j) - \frac{1}{\Delta s}(-\lambda_i^j + Ad^T_{\exp(\Delta s\xi_i^{j-1})}\lambda_i^{j-1}) + \bar{W}_{i-1}^{j-1} \tag{3.36}$$

where we define the discrete conjugate momenta $\mu_i^j = \operatorname{dexp}^{-T}_{\Delta t\eta_i^j} M\eta_i^j$ and $\lambda_i^j = \operatorname{dexp}^{-T}_{\Delta s\xi_i^j} K\Delta\xi_i^j$. μ can be viewed as the momentum of the body and λ as the internal stress on the body. To complete the scheme we can relate ξ and η together using multiple applications of (3.26) and (3.27) and eliminating the g's.

$$\exp(\Delta t\eta_i^{j-1})\exp(\Delta s\xi_{i+1}^{j-1}) = \exp(\Delta s\xi_i^{j-1})\exp(\Delta t\eta_i^j) \tag{3.37}$$

Using (3.27), (3.36), (3.37), and the definitions of μ and λ one time step of the integrator is defined by:

$$\xi_{i+1}^j = \frac{1}{\Delta s}\log(\exp(-\Delta t\eta_i^j)\exp(\Delta s\xi_i^j)\exp(\Delta t\eta_i^{j+1})) \tag{3.38}$$

$$g_{i+1}^{j+1} = g_{i+1}^j\exp(\Delta s\hat{\xi}_{i+1}^j) \tag{3.39}$$

$$\lambda_{i+1}^j = \operatorname{dexp}^{-T}_{\Delta s\xi_{i+1}^j} K\Delta\xi_{i+1}^j \tag{3.40}$$

$$\mu_{i+1}^j = Ad^T_{\exp(\Delta t\eta_i^j)}\mu_i^j - \frac{\Delta t}{\Delta s}(-\lambda_{i+1}^j + Ad^T_{\exp(\Delta s\xi_{i+1}^{j-1})}\lambda_{i+1}^{j-1}) + \Delta t\bar{W}_i^{j-1} \tag{3.41}$$

$$\mu_{i+1}^j = \operatorname{dexp}^{-T}_{\Delta t\eta_{i+1}^j} M\eta_{i+1}^j \tag{3.42}$$

where all the equations are explicit except for (3.42) which is used to implicitly solve for η_{i+1}^j making this scheme semi-implicit. We note that a first order approximation of dexp can be used and still preserve the accuracy which makes it possible to solve (3.42) easily using Newton iteration.

In fact, typically the Newton iteration converges within 2 steps to a tolerance of 1×10^{-14} which effectively makes this semi-implicit scheme fully explicit in terms of computation time given the quickness of convergence. In this scheme it is necessary to compute the tip ξ from the tip boundary condition and to specify the g and η at the base.

Due to the variational integrator being effectively explicit it is quite efficient to compute a time step; however, due to the system being hyperbolic it is heavily restricted by the Courant-Friedrichs-Lewy (CFL) condition [65]. The CFL condition essentially says that the space and time discretizations need to be the proper resolution to propagate a wave at most one spatial step at each time step, which can be stated as:

$$\frac{\Delta s}{\Delta t} \geq c \tag{3.43}$$

where c is the wave speed for the material. The wave speed can be approximated as $c = \sqrt{\frac{E}{\rho}}$ for the rod. Using a factor of safety we can compute the time step from the spatial discretization and the material properties as:

$$\Delta t = 0.1 \Delta s \sqrt{\frac{\rho}{E}} \tag{3.44}$$

where 0.1 is the factor of safety determined from testing the simulation stability. The result is that smaller space discretizations and stiffer materials cause the maximum time step to decrease. Unfortunately, this restricts the ability of the simulation to being real-time only for small numbers of discretizations (e.g., < 20) and for soft materials (e.g., Ecoflex). After implementing the simulation in C using the GNU Scientific Library (gsl) for the matrix and vector manipulation the simulations can run in real-time under the discussed restrictions, the details of the timing will be considered further in the control section.

3.3.2 Kirchhoff Restriction

The CFL condition is a difficult restriction to overcome especially when the simulated system is numerically stiff [65]. Numerical stiffness of a system comes from the transient components of the system vibrating at different frequencies and the bigger the difference the more stiffness that is

Given: g_i, ξ_i, η_i, μ_i, λ_i ;
Base conditions: $g^0_{i+1} = I_{4\times4}$, $\eta^0_{i+1} = 0$, $\mu^0_{i+1} = 0$;
Tip conditions: $\xi^N_{i+1} = K^{-1}W_{ext} + \xi^*$, $\lambda^N_{i+1} = \text{dexp}^{-T}_{\Delta s\xi^N_{i+1}} K\Delta\xi^N_{i+1}$;
Set: $\xi^0_{i+1} = \frac{1}{\Delta s}\log(\exp(-\Delta t\eta^0_i)\exp(\Delta s\xi^0_i)\exp(\Delta t\eta^1_i))$;
$\lambda^0_{i+1} = \text{dexp}^{-T}_{\Delta s\xi^0_{i+1}} K\Delta\xi^0_{i+1}$;
for *j=1..N-1* **do**
  $\xi^j_{i+1} = \frac{1}{\Delta s}\log(\exp(-\Delta t\eta^j_i)\exp(\Delta s\xi^j_i)\exp(\Delta t\eta^{j+1}_i))$;
  $\lambda^j_{i+1} = \text{dexp}^{-T}_{\Delta s\xi^j_{i+1}} K\Delta\xi^j_{i+1}$;
  $g^j_{i+1} = g^{j-1}_{i+1}\exp(\Delta s\hat{\xi}^{j-1}_{i+1})$;
  $\mu^j_{i+1} = Ad^T_{\exp(\Delta t\eta^j_i)}\mu^j_i - \frac{\Delta t}{\Delta s}(-\lambda^j_{i+1} + Ad^T_{\exp(\Delta s\xi^{j-1}_{i+1})}\lambda^{j-1}_{i+1}) + \Delta t\bar{W}^{j-1}_i$;
  Solve η^j_{i+1}: $\mu^j_{i+1} = \text{dexp}^{-T}_{\Delta t\eta^j_{i+1}} M\eta^j_{i+1}$;
end
Set: $g^N_{i+1} = g^{N-1}_{i+1}\exp(\Delta s\hat{\xi}^{N-1}_{i+1})$;
$\mu^N_{i+1} = Ad^T_{\exp(\Delta t\eta^N_i)}\mu^N_i - \frac{\Delta t}{\Delta s}(-\lambda^N_{i+1} + Ad^T_{\exp(\Delta s\xi^{N-1}_{i+1})}\lambda^{N-1}_{i+1}) + \Delta t\bar{W}^{N-1}_i$;
Solve η^N_{i+1}: $\mu^N_{i+1} = \text{dexp}^{-T}_{\Delta t\eta^N_{i+1}} M\eta^N_{i+1}$;

Algorithm 1: Variational dynamics simulation

present. Stiffness causes an issue because all the waves need to be resolved properly to be stable, but often the small and fast vibrations are insignificant to the overall behavior. For the Cosserat rod the linear deformations (shear and extension) are far stiffer than the angular deformations (bending and torsion) causing the linear deformations to vibrate much faster than the angular ones. The stiffness comes from the diameter, D, being relatively small for a rod. Since the linear stiffness is proportional to D^2 and the angular stiffness proportional to D^4 the linear stiffnesses can be significantly larger. However, the linear deformations are typically insignificant to the overall behavior of the rod which means the simulations are being restricted to small time steps because of insignificant phenomena. Since in some cases we can ignore the linear deformation we could benefit by having a less restrictive CFL condition in the simulations. When it is possible to ignore linear deformations we can use a Kirchhoff rod model. The Kirchhoff rod is the same as the Cosserat rod except it is assumed that there is no shear or extension. The reduction is made simply by:

$$\nu = \nu^* \tag{3.45}$$

Which means the linear portion of the spatial twist never changes. The relevant changes to the dynamic equations are rather minimal, but a tip boundary condition only deals with moments as only the angular portion of ξ can vary.

Making this simplification in the simulations is fairly straightforward. It does not influence the running time much since most of the same computations are still carried out, but the safety factor greatly increases in the time step. We found that the safety factor can be increased from 0.1 to 0.9 and will remain stable due to the reduced numerical stiffness. The Kirchhoff simplification loses some qualities of the behavior, primarily the extension, but they are likely minimal, especially with stiffer materials, and significantly improve the potential to reach real-time simulations and control.

3.3.3 Implicit Schemes

While it is possible to achieve real-time simulations with the presented variational integrators the CFL condition for the explicit schemes is very limiting for stiff materials and fine spatial discretizations. One way to avoid the CFL condition is to use implicit time discretization schemes [65]. The CFL condition comes from the need for propagating waves to be properly resolved. In a numerical simulation for a propagating wave the peak of the wave cannot skip past a spatial point or else it will be unstable, so to guarantee that a wave moves at most one spatial point in a given time step the CFL condition must be met [66]. This can be seen as the propagation of information in the system is restricted to the wave speed. The reason why an implicit scheme avoids the CFL condition is because all the information at every point in the domain is available to every other point meaning there is effectively no limit on how fast the information can move in the system. This means there is no CFL condition for stability and larger time steps can be taken. However, implicit schemes generally need to solve a large system of equations which is quite an expensive computation limiting the utility of the scheme for real-time implementations. Recently, it was noticed that for soft manipulators if the PDEs are first discretized implicitly in time to get ODEs in space the equation solving can be simplified to just one equation [40] rather than an equation at every spatial point. This approach gets the benefit of avoiding the CFL restriction and stays

fairly computationally efficient due to only having to solve one equation, or 6 components of one vector equation. This presents a possibly large benefit for real-time simulations and control.

The mentioned implicit scheme works by implicitly discretizing in time to get ODEs in space and then a shooting method is used to integrate the ODEs in space and solve the boundary conditions. One time step is similar to the statics if the inertia terms are viewed as an external load. For this approach it is easy to preserve the Lie group structure through the RKMK technique; however, preserving energy isn't as straightforward. Due to the explicit integration in space there will possibly be some energy dissipation in the integration, but it will be quite small considering that the integration will be over a relatively short domain (the length of the rod). So, the dissipation in space is negligible and that means the time scheme needs to preserve energy if we want the simulation to be symplectic like the variational integrators were.

It turns out that the implicit scheme observation is not compatible with the discussed variational integrator, as a full system of equations is always needed to be solved for rather than a single equation when going through the derivation. Luckily there are several kinds of integrators that meet the requirement of being implicit and symplectic that can be used instead of the variational integrators. For example all integrators using Gauss-Legendre quadrature are implicit and symplectic [59]. The simplest Gauss-Legendre scheme is the implicit midpoint rule, $x_{i+1} = x_i + \Delta t \dot{x}_{i+1/2}$.

Using this approach we will make a simulation that uses the implicit RKMK midpoint rule in time and an explicit RKMK scheme in space. To begin we can restate the important dynamic equations:

$$(-\frac{d}{dt} + ad_\eta^T)\mu - (-\frac{d}{ds} + ad_\xi^T)\lambda + \bar{W} = 0 \tag{3.46}$$

$$\dot{\xi} = (\frac{d}{ds} + ad_\xi)\eta \tag{3.47}$$

$$\mu = M\eta \tag{3.48}$$

$$\lambda = K\Delta\xi \tag{3.49}$$

where μ and λ are the conjugate momenta which correspond to momentum and stress respectively. Using the implicit midpoint rule we approximate the time derivatives as:

$$\dot{x}_{i+1/2} = \frac{x_{i+1} - x_i}{\Delta t} \tag{3.50}$$

where the subscripts denote the time step. The half denotes being halfway between the next and previous time steps. In order to stay consistent the ODEs in space will be evaluated at the half step as:

$$\lambda'_{i+1/2} = \frac{\mu_{i+1} - \mu_i}{\Delta t} - ad^T_{\eta_{i+1/2}}\mu_{i+1/2} + ad^T_{\xi_{i+1/2}}\lambda_{i+1/2} - \bar{W}_{i+1/2} \tag{3.51}$$

$$\eta'_{i+1/2} = \frac{\xi_{i+1} - \xi_i}{\Delta t} - ad_{\xi_{i+1/2}}\eta_{i+1/2} \tag{3.52}$$

In order to integrate these equations we need to know the initial conditions. The initial condition for η is known since the base is assumed fixed; however, we do not know ξ because we only have a condition for ξ at the tip, $K\Delta\xi = W_{ext}$. So, just like the statics the spatial ODEs can be integrated using a shooting method where $\xi(0)$ is guessed, the system integrated, and then $K\Delta\xi(L) = W_{ext}$ checked to see if the system is solved. This describes how the system is implicit in only one equation, the tip condition.

The half step values can be solved for using a shooting method, but those need to be used to step forward in time. In order to use the half step values to step forward in time they need to be related to the next time step. This can be done by considering the half step to just be the average between the previous and next time steps (e.g., $\eta_{i+1/2} = \frac{\eta_{i+1}+\eta_i}{2}$). This allows for the next time step values to be computed from the half step values. After integration, stepping forward in time is done by:

$$\eta_{i+1} = 2\eta_{i+1/2} - \eta_i \tag{3.53}$$

$$\xi_{i+1} = 2\xi_{i+1/2} - \xi_i \tag{3.54}$$

Using the explicit RKMK Euler scheme to integrate in space and the implicit RKMK midpoint scheme in time, the RKMK in time is for stepping g, we get a symplectic Lie integrator for the manipulators that can take long time steps. It is important to note that implicit schemes rather than having a maximum time step instead have a limit on how short the time step can be. This is due to dividing by small numbers causing ill-conditioning of the equations in the equation solving step. This lower bound is usually found empirically [40]. The computational efficiency will be discussed further in the control section.

Given: g_i, ξ_i, η_i ;
Assume the $\xi^0_{i+1} = \xi^0_i$;
while *Tip condition not solved* **do**
  Integrate with explicit Euler RKMK:
  $\lambda'_{i+1/2} = 2\frac{\mu_{i+1/2}-\mu_i}{\Delta t} - ad^T_{\eta_{i+1/2}}\mu_{i+1/2} + ad^T_{\xi_{i+1/2}}\lambda_{i+1/2} - \bar{W}_{i+1/2}$;
  $\eta'_{i+1/2} = 2\frac{\xi_{i+1/2}-\xi_i}{\Delta t} - ad_{\xi_{i+1/2}}\eta_{i+1/2}$;
  $g'_{i+1/2} = g_{i+1/2}\hat{\xi}_{i+1/2}$;
  Set:
  $\xi_{i+1} = 2\xi_{i+1/2} - \xi_i$
  $\eta_{i+1} = 2\eta_{i+1/2} - \eta_i$
  $g_{i+1} = g_i \exp(\Delta t \eta_{i+1/2})$;
  compute boundary condition error;
  update ξ^0_{i+1} by root finding method;
end

Algorithm 2: Implicit dynamics scheme

3.4 Computational Results

To compare the behavior of the schemes we try the simple cases of a cantilever rod vibrating with gravity and a cable driven manipulator responding to a sequence of step inputs. Due to similarities in the responses we compare the explicit and implicit Cosserat dynamics for the cantilevered rod and the explicit Cosserat and Kirchhoff in the cable driven manipulator.

For the cantilevered rod we look at the response with some viscosity ($\alpha = 300$ Pa·s) for 10s. We compare the convergence due to adjusting the spatial discretization and the time steps. When increasing the number of spatial steps for the integration we see that the amplitude of the vibrations

decrease, Figure 3.3, effectively showing that increasing the size of the spatial steps softens the body. By adjusting the number of nodes we see that convergence to a consistent amplitude happens around $N = 20$. For the time discretization we see that in the implicit scheme that at larger time steps there is an extra secondary periodic wave present in the vibration that should not be there, but as we decrease the time step this goes away, around $\Delta t = 0.01$s is when this becomes unnoticeable, Figure 3.4. We also note that the number of spatial points in the implicit scheme is an important consideration for lowering the time step, for example the approximate lower bound on the time step for the implicit scheme for $N = 20$ is $\Delta t = 0.04$s. Here we also show a comparison in the response for the cantilever between implicit and explicit schemes for $N = 20$ and the implicit time step is $\Delta t = 0.05$s. We see that the amplitudes and frequencies are the same, but given the different resolutions the responses are slightly different, Figure 3.5

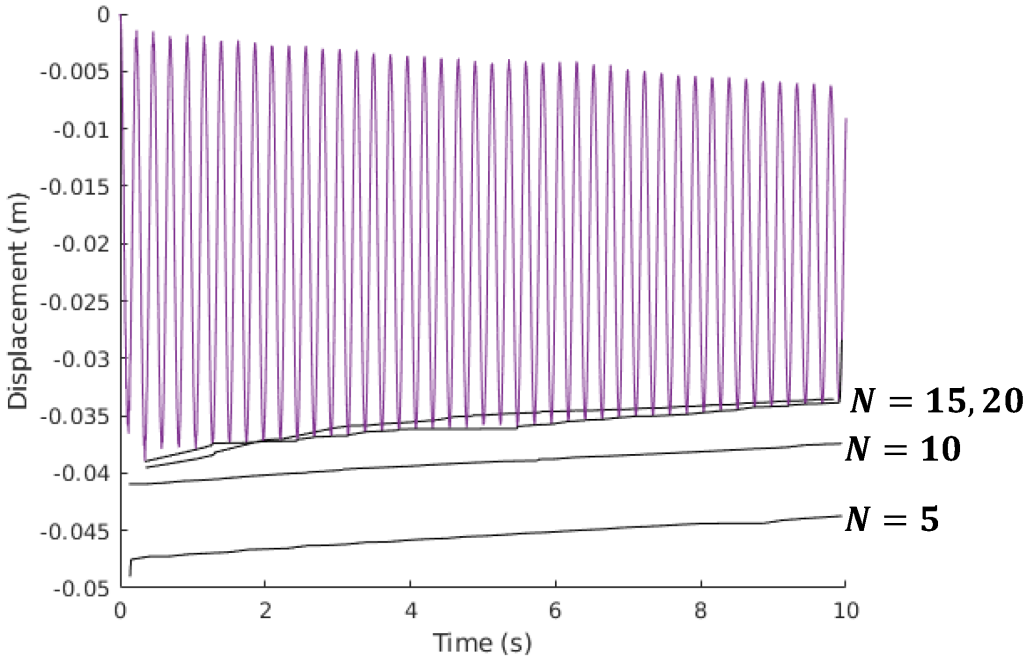
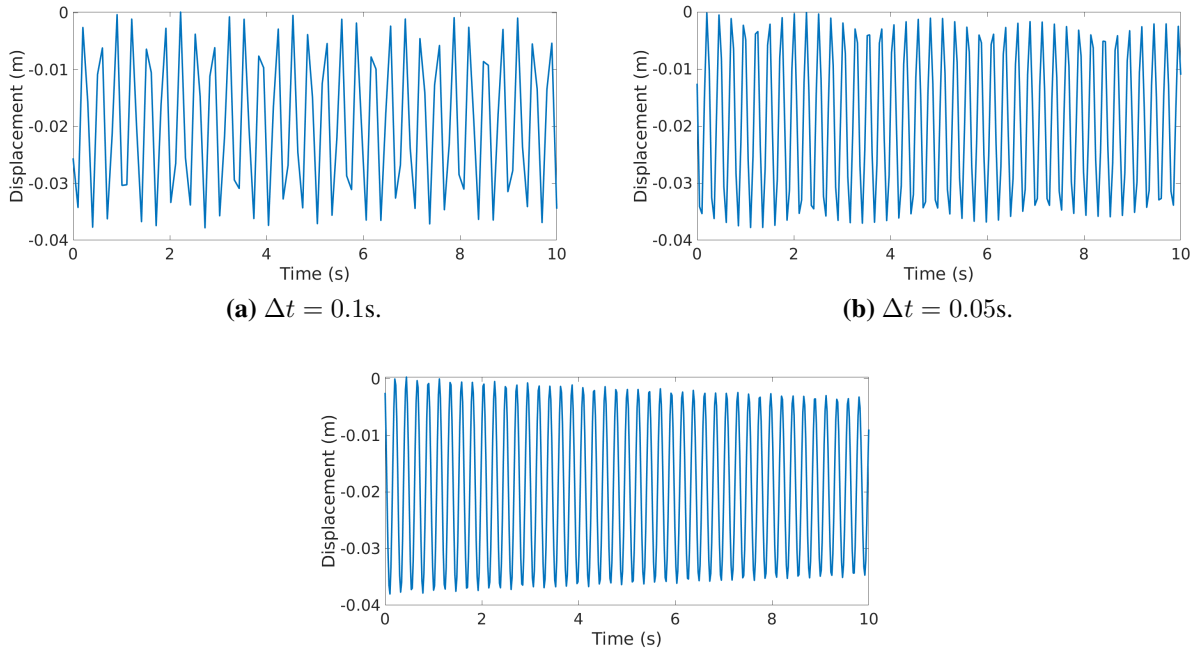


Figure 3.3: Using the explicit variational Cosserat we show the convergence of the amplitude with respect to the number of spatial points used. Here $N = 5, 10, 15, 20$ are shown and we see convergence around $N = 15$ to $N = 20$ after $N = 20$ changes are insignificant. Only the lower asymptote is shown as the upper asymptote does not change much.



(a) $\Delta t = 0.1$s.

(b) $\Delta t = 0.05$s.

(c) $\Delta t = 0.02$s.

Figure 3.4: The time discretization leads to errors in the period of the wave, we see multiple different waves rather than a single vibration. Using the implicit integrator we show how the spurious waves disappear with smaller discretization. All used 100 spatial points.

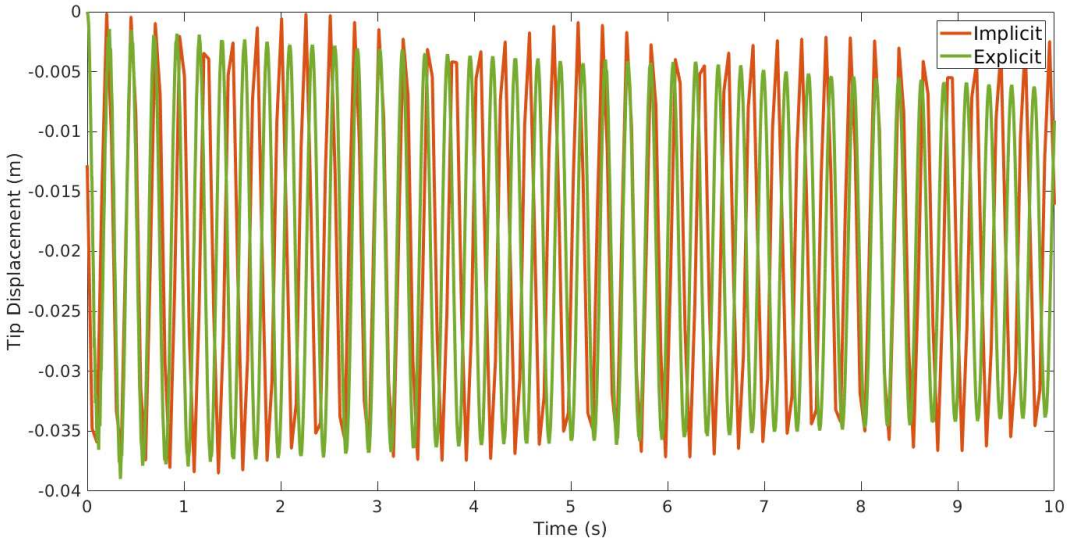


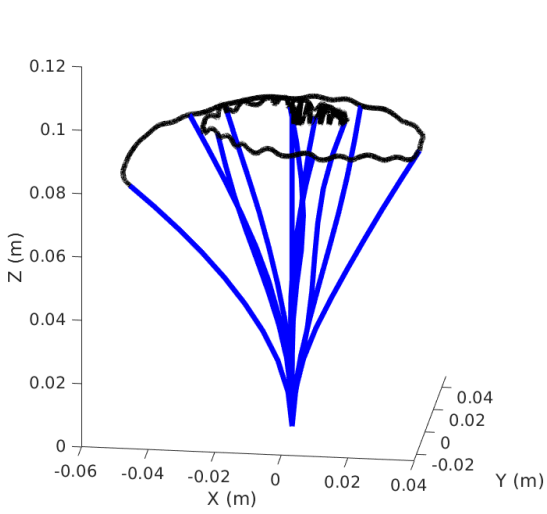
Figure 3.5: Shows the dynamic response by recording the tip displacement through time for the implicit and explicit Cosserat schemes. Uses 20 spatial points for both simulations, for the implicit simulation a $\Delta t = 0.05$s is used as much lower leads to numerical ill-conditioning for this amount of spatial points.

Responding to cable actuation what we see is something somewhat surprising, the trajectories of the tip between the Cosserat and Kirchhoff models end up being significantly different, Figure 3.6. The main difference is that the Cosserat rod vibrates in the axial direction when responding to the change in actuation and this slows down the overall motion while the Kirchhoff rod does not vibrate anywhere near as much and responds much quicker. This shows that the Kirchhoff model can become inaccurate depending on the system being modelled, typically it isn't as good for softer materials and actuation that applies large forces axially. This also reveals a significant consideration in the control of the rods, the extra axial vibrations in the Cosserat rod will result in control being more difficult as discussed in the control later.

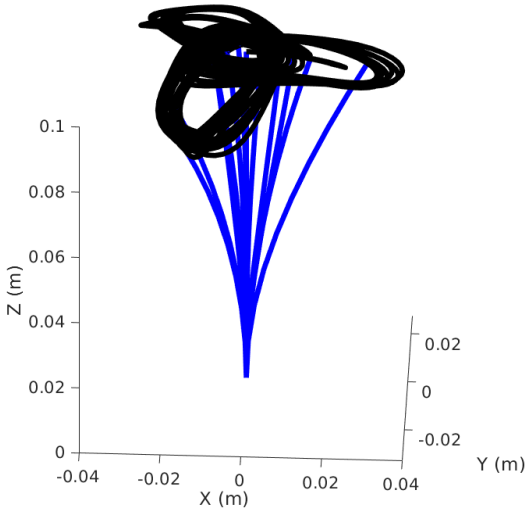
3.4.1 Dynamic Workspace

Like traditional robots, soft robots have a workspace to represent its possible configurations; however, the soft robots had a few extra considerations as discussed in the Statics. As a result we used an approximate form of the rod to characterize the workspace. For the dynamics we have a new consideration that is not present in traditional robots. Soft robots can reach outside their static workspace by taking advantage of their flexibility and using dynamic motions. This has interesting implications in terms of control as building up momentum is a valid strategy for reaching desired points, but these points cannot be held. So, we need to characterize the dynamic workspace as well as the static workspace to differentiate between points that can be reached and held and only those that can be reached.

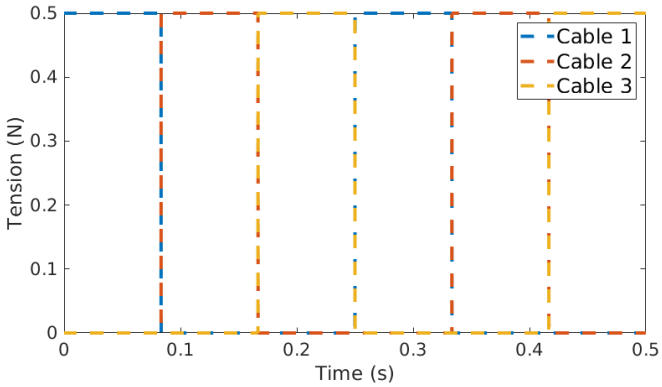
For the representation of the workspace we keep the same form as the statics. This form seems to be a reasonable estimate for the workspace. In order to sample the workspace we do not have a guaranteed way of determining representative states, so we instead give random inputs to the simulation and determine the workspace that encapsulates all the points. The inputs are held constant for several steps so that momentum can be built up. The resulting workspace can be seen in Figure 3.7 and a comparison between static and dynamic can be seen in Figure 3.8.



(a) Cosserat response.



(b) Kirchhoff response.



(c) Cable tensions.

Figure 3.6: The dynamic responses for changing step inputs for a cable manipulator modelled as a Cosserat rod (left) and a Kirchhoff rod (center). The blue lines represent the centerline of the rod and the black line tracks the tip position. The cable tensions are a series of step inputs shown on the right.

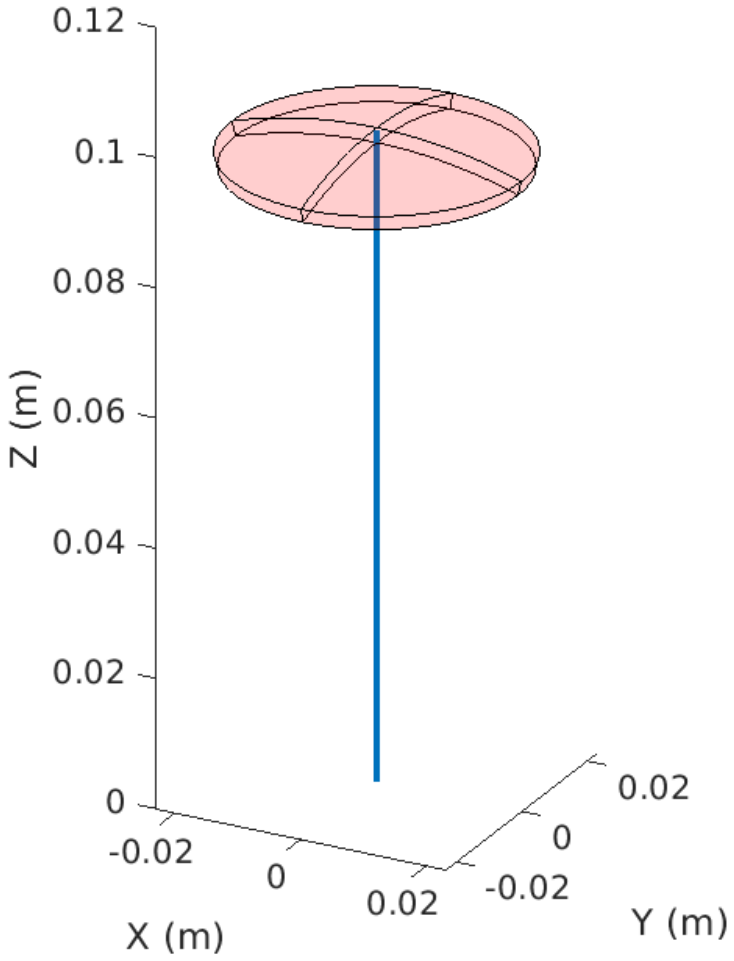


Figure 3.7: Show the dynamic workspace for the cable driven manipulator with the tension limited to 0.5 N.

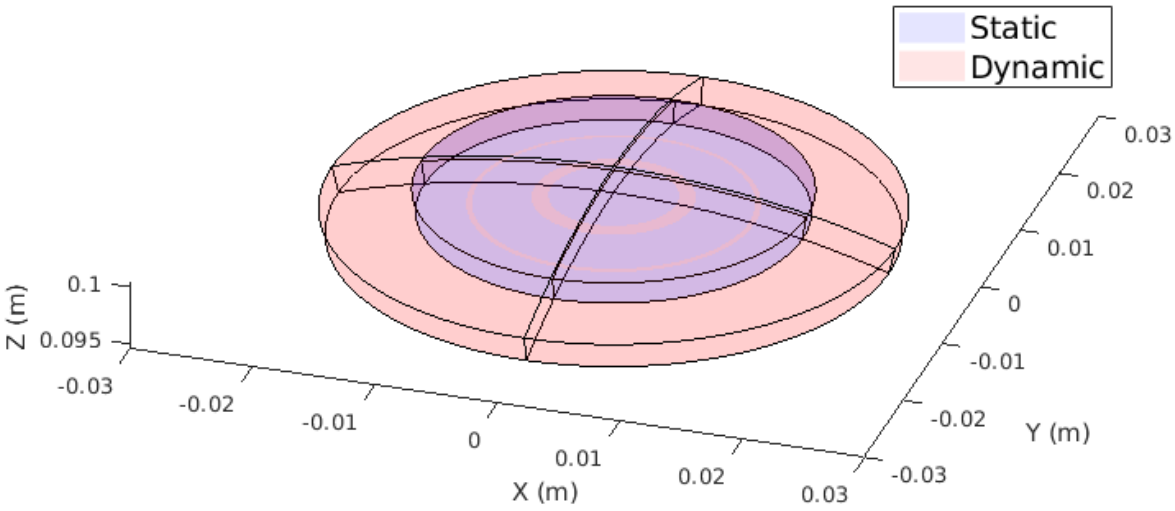


Figure 3.8: Compares the static (blue) and dynamic (red) workspaces.

3.5 Summary

In this chapter we extended the statics to the dynamics by applying Hamilton's principle, considered some new loading cases, and developed the numerical simulations for the dynamics. For the dynamics Hamilton's principle allowed us to extend the principle of virtual work from the statics to the dynamics by including the kinetic energy. Due to the dynamic motion we now have dynamic effects such as viscosity and the dynamics of the embedded actuators. For the numerical simulations we discussed how we want to eliminate numerical damping and continue to preserve the Lie group structure as was done in the statics. For this we implement a variational integrator for the dynamics, but due to the CFL condition we are heavily restricted in the size of the time step. Then, we present an implicit scheme that has no numerical damping, preserves the Lie group structure, and avoids the CFL condition to compare with the variational integrator.

Chapter 4

Soft Manipulator Control

The goal of control is to determine the inputs to the soft manipulator that will achieve a certain objective. In general, control can be quite a hard problem to address and for soft robots this holds true. Controlling soft robots suffers from two primary problems: the configurations have infinite degrees of freedom and the computational complexity of the models. There are a variety of possible control tasks, but the simplest one which will be looked at is the control of the tip position of the manipulator.

An important consideration for soft robots is the existence of both a static and dynamic workspace. Tasks defined in the static workspace are fundamentally different from those in the dynamic workspace as static points can be held stably, while dynamic points can only be reached temporarily. Soft robots are also inherently underactuated, there are more degrees of freedom than inputs, making the dynamics a much more important feature of the control [49].

The general framework for conceptualizing a control problem is to find the actions that minimize the total cost related to the objective. The actions are also subject to the dynamics of the system and the constraints imposed on the system. This can be stated as:

$$\min_{u} \qquad J = C_{end}(x_f) + \int_{t_0}^{t_f} C(x, u, t)dt \tag{4.1}$$

$$\text{subject to: } \dot{x} = f(t, x, u) \tag{4.2}$$

$$B(x_0, x_f) = 0 \tag{4.3}$$

$$A(x, u) \leq 0 \tag{4.4}$$

where J is the total cost, C_{end} is the cost of the end point, C is the one step cost, f is the system dynamics, B are the boundary conditions, A are other system constraints, x are the state variables,

and u are the inputs to the system. In general solving this is not a simple task and usually involves either solving a nonlinear programming problem or a complex boundary value problem [49].

Due to the difficulty of optimal control problems many techniques exists for solving them [49]. Here we will go over some approaches to controlling the soft manipulator. We will start with a simple quasi-static approach, move to Model Predictive Control (MPC) approaches, and then turn to machine learning techniques.

Throughout all the control tests the programs are ran on a single thread of i7-2.10GHz CPU with 8GB ram, the simulations are implemented in C, the optimization based control schemes are implemented in Matlab, and the learning algorithms are implemented in python. Time has been taken to optimize the simulations a bit, but the other algorithms are not necessarily optimized for time efficiency.

4.1 Quasi-static Control

Typically control problems deal with dynamic systems; however, if we assume the system to undergo quasi-static motion we can pose a control problem as solving the inverse statics [43]. The inverse statics is given a desired steady state tip configuration solve for the actuator forces.

To start, the system of equations for the statics can be summarized as:

$$g' = g\hat{\xi} \tag{4.5}$$

$$\xi' = f(s, q, g, \xi) \tag{4.6}$$

$$b(q, g, \xi) = 0 \tag{4.7}$$

where q is the actuation inputs, f is the statics ODE, and b stands for the boundary conditions. If we then define the error, $e(q, \xi_0)$, we can determine an iterative scheme for finding $e = 0$.

To start, the differential of e is:

$$\begin{aligned} de &= \frac{de}{dq}dq + \frac{de}{d\xi_0}d\xi_0 \\ &= E_q dq + E_\xi d\xi_0 \end{aligned} \tag{4.8}$$

where E_q and E_ξ are the error jacobians.

Also evaluating the differential of the boundary condition gives:

$$\begin{aligned} db &= \frac{db}{dq}dq + \frac{db}{d\xi_0}d\xi_0 \\ &= B_q dq + B_\xi d\xi_0 \end{aligned} \tag{4.9}$$

where B_q and B_ξ are the boundary condition jacobians.

Due to the boundary always needing to be satisfied we know that $db = 0$ which allows us to rearrange to get:

$$\begin{aligned} de &= (E_q - E_\xi B_\xi^{-1} B_q)dq \\ &= J dq \end{aligned} \tag{4.10}$$

where J is the overall jacobian.

Using (4.10) we could determine the q necessary to make $e = 0$; however, analytic forms for the jacobians generally won't exist so it is necessary to discretize (4.10).

$$\Delta e = J \Delta q \tag{4.11}$$

where the components of J (E_q, E_ξ, B_q, and B_ξ) are computed by perturbing a known solution to the statics and approximating the jacobians using finite differences. This gives an iterative scheme for determining q, $\Delta q = J^{-1}\Delta e$. Note that generally a gain factor needs to included into the update, $q_{i+1} = q_i - 0.2 J_i^{-1} e_i$, to remain stable when the desired point is far from the current point.

This essentially gives a quasi-static control scheme where it is assumed that dynamic effects are negligible as the q can be updated when the objective changes or iteratively updated to get the desired configuration.

To test the effectiveness of this scheme we see how many steps it takes to get within 1% tip error of a randomly selected point in the static workspace. We also see how effectively it tracks a circular trajectory in the static workspace.

For the reaching test we do 100 randomly sampled points and record the number of steps it takes to reach them. The mean number of steps was 9.33 and the standard deviation was 3.24. The sampled points can be seen in Figure 4.1.

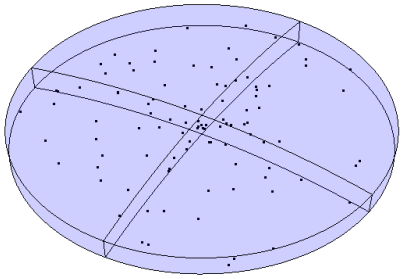


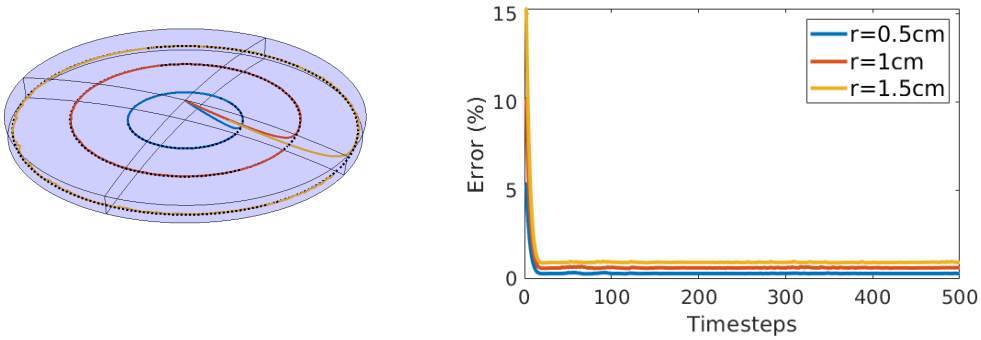
Figure 4.1: The dots represent the randomly sampled points to be reached within the static workspace.

For the tracking we attempt to track 3 different circles with different radii. The radii are 0.5 cm, 1 cm, and 1.5 cm with the heights selected so that all the points are within the static workspace. In general the tracking error was quite low, less than 1% for all circles, and the reason it is higher for the larger radii seems to be due to the sweeping speed of the larger circle being a bit faster and therefore a bit harder to keep up with.

It seems unlikely that soft manipulators will act quasi-statically, but it provides a starting place and a comparison.

4.2 Model Predictive Control

One of the main concerns with controlling the soft manipulator is the computational cost of the simulations. Having simulations that are computationally expensive makes achieving real-time



(a) Quasi-static Tracking Paths. **(b)** Quasi-static Tracking Errors.

Figure 4.2: The quasi-static tracking of the circular paths. The manipulator starts at its reference and tracks the circle. The dashed black lines represent the desired circles.

control is difficult because each evaluation of the simulations will slow down the optimization routine. In general a global solution to a problem will be impractical to compute, but a local solution is viable. Local solutions typically work by looking over a finite time horizon and optimizing the trajectory over that horizon, taking the initial step, and then redoing the optimization again. It is also common to make simplifications to the local cost and the local dynamics of the system to further improve computation speed. Another concern is the state space, a soft manipulator has infinite degrees of freedom and so needs a finite representation of the state space. Then, the finite state space can still be fairly large making optimization more difficult. In the simulations there is also the trade-off that more accurate representations are slower to compute and can restrict the length of the time steps taken. So, the considerations to balance with the control problem are: the time horizon in the control process, the resolution of the state space, and the resolution of the simulation.

One popular approach for control is Model Predictive Control (MPC). The idea behind MPC is to optimize the trajectory of the system over some finite time horizon and then to command the initial action of that trajectory, once the step is taken the optimization is done again with the time horizon shifted forward one step. MPC does not give a global optimal solution, but it gives a

local optimal solution which is generally satisfactory with less computational burden and an easier ability to integrate it into a real-time system.

At a time step the MPC routine can be described as:

$$\min_{u} \qquad J(x,u) = \sum_{i=0}^{N} C(x_i, u_i, t_i) \tag{4.12}$$

$$\text{subject to: } x_{i+1} = f(x_i, u_i, t_i) \tag{4.13}$$

where N is the number of time steps to look into the future, x_0 is the current state, f is the discrete dynamics, and extra constraints are not considered. Once the system is solved u_0 is taken, the system is measured, and then the process repeats shifted one time step into the future. In general for a nonlinear problem the optimization can be done by techniques such as least squares fitting.

One thing that can be done in the finite horizon with discrete steps is to formulate the problem as a dynamic programming problem. This starts by defining the cost-to-go function, V, as:

$$V(x,i) = \min_{U} J(x,U) \tag{4.14}$$

which makes the cost-to-go the accumulated cost of the trajectory after step i. Using $V(x,N) = C(x_N)$, since the final state cannot accumulate more future cost, the cost-to-go can be restated as:

$$V(x,i) = \min_{u} C(x,u) + V(f(x,u), i+1) \tag{4.15}$$

which is the Bellman equation [67]. The significance of the Bellman equation is that the optimization now occurs at only the current state rather than all states simultaneously. However, what V actually is can be difficult to compute leading to other techniques of optimization and approximation to determine V. The Bellman equation also shows up in machine learning techniques, but it is more typically phrased as a value function where rewards instead of costs are accumulated.

The main restriction when trying to apply control techniques is whether or not the computation is real-time. For something to be real-time the computation of the control command must be faster

than the time step of the system. First, we can quantify the real-time performance of the simulation itself, using the explicit Cosserat scheme as the example. If we define $T(n)$ as the time it takes to do one step of the integration with n spatial points then we can define the real-time ratio, τ_{sim}, as:

$$\tau_{sim} = \frac{T(n)}{\Delta t} = \frac{T(n)\sqrt{E}(n-1)}{0.1L\sqrt{\rho}} \tag{4.16}$$

where Δt is the time step of the simulation, E is the elastic modulus of the rod, L is the length of the rod, and ρ is the density of the rod. The value 0.1 is an empirical safety factor for stability of the simulation. The simulation is then real-time if $\tau < 1$ and soft real-time when $\tau = 1$ (real-time, but noise may make some steps lag behind). It is apparent that achieving real-time is much easier when the material is softer (wave speed is lower) and when the spatial resolution is low.

For the control scheme to be real-time the optimization time must be less than the one-step command. This can be seen as evaluating many steps of the simulation and then acting on only one, which makes the real-time ratio of the control scheme, τ, approximately:

$$\tau \approx m\tau_{sim} \tag{4.17}$$

where m is the rough number of evaluations of the simulation the optimization routine needs to do. Other computations need to be involved in the optimization, but the main computational cost is the simulation. Then, if the number of steps in the trajectory is N we get $m = NS$ where S are the number of sampled trajectories. This will give a rough estimate of the possible time horizons and the iterations of the optimization that can be done.

To do some reasoning about real-time factors of the control schemes we first measure $T(n)$. The result is that the computation time is nearly linear with n and is roughly $T(n) = 4.9 \times 10^{-5} + 6.5 \times 10^{-6}n$ seconds over the range $n = 10, \ldots, 100$ on a single thread of an i7-2.10GHz CPU for the explicit schemes. This shows that the individual steps are fairly fast to compute, but the CFL condition is still quite restrictive over the material properties. The real-time ratio for the simulation can be approximated as:

$$\tau_{sim} = \frac{1}{L}\sqrt{\frac{E}{\rho}}(-4.9 \times 10^{-4} + 4.25 \times 10^{-4}n + 6.5 \times 10^{-5}n^2) \tag{4.18}$$

With the known material properties we can determine the number of spatial points available to get a desired real-time ratio. Soft real-time essentially gives an upper bound to the number of spatial steps that can be used for the given material properties. At soft real-time, $\tau_{sim} = 1$, we get the value for n as:

$$n = \frac{-4.25 \times 10^{-5} + \sqrt{3.08 \times 10^{-9} + 2.6 \times 10^{-6} L\sqrt{\frac{\rho}{E}}}}{1.3 \times 10^{-5}} \tag{4.19}$$

where the negative root is removed to keep time positive. If we select $L = 10cm$ and $\rho = 1000\frac{kg}{m^3}$ then we can see the relationship between material stiffness and the number of spatial points available. We can see that $n \approx 10$ around $E \approx 100kPa$ and $n \approx 5$ around $E \approx 1MPa$, which ends up being quite restrictive on the control algorithm and simulations. So, the predictive control will likely be leaving a lot to be desired, but it can still be performed.

However, this assumed that the scheme was the explicit Cosserat scheme. If the Kirchhoff restriction is made the time steps become nine times larger and the result is that the soft real-time time steps become:

$$n_{Kirchhoff} = \frac{-4.25 \times 10^{-5} + \sqrt{3.08 \times 10^{-9} + 2.34 \times 10^{-5} L\sqrt{\frac{\rho}{E}}}}{1.3 \times 10^{-5}} \tag{4.20}$$

which is a bit less restrictive as $n \approx 34$ at $E \approx 100kPa$ and $n \approx 18$ at $E \approx 1MPa$. So, the benefit of using the Kirchhoff simplification is clear in terms of control.

For the implicit scheme the considerations of timing and stability are slightly different. Implicit schemes are not subject to the CFL condition, so there is no upper bound on the time step; however, implicit schemes can become unstable at smaller time steps. The point of instability is determined empirically. Due to needing to solve the boundary condition the implicit scheme naturally takes longer than the explicit schemes per time step. To get the approximate running time with spatial

discretization we use the simulation with no viscosity. The time relationship is: $T(n) = -2 \times 10^{-4} + 9.87 \times 10^{-5} n$, which is roughly 15 times slower than the explicit scheme, but we have no upper bound on the time step making real-time simulations more easily achievable. So, for the implicit scheme the real-time ratio is:

$$\tau_{sim} = \frac{-2 \times 10^{-4} + 9.87 \times 10^{-5} n}{\Delta t} \tag{4.21}$$

If a particular n or Δt is desired then the other can be determined to achieve real-time.

The explicit schemes seem to generally be fairly restricted due to the CFL condition for usage in control, so the implicit scheme likely has the advantage. The only downside being if we want a small time step the implicit scheme can become unstable and the explicit schemes are more desirable, though small enough time steps to become unstable are not going to be used in our tests.

4.2.1 Nonlinear Least Squares

For MPC there are many techniques for computing the optimal trajectory, one such method is Nonlinear Least Squares (NLSQ). NLSQ is a general algorithm for finding local optima for nonlinear problems when they have an error that can be written as a sum of squares. Since we can specify the cost for the manipulator tasks as a Euclidean norm this is a plausible technique for approaching the control problem. However, one benefit is that the objective does not need to be specified over the whole state and can instead be only specified in the task space. The task space consists of the errors in the parts of the state trying to be controlled, so in our case the error in the tip position. The benefit of this is that approximations to gradients and jacobians are much less expensive because instead of trying to find the optimal input by looking at the whole change in state it only finds the optimal inputs for the change in the task space.

The way NLSQ is used for controlling the soft manipulators it to solve for the optimal sequence of inputs that minimize the total cost of the sequence of states in the time horizon. When given the desired tip position the cost would be the sum of the tip position errors measured at each time step,

$\sum \|p - p_{des}\|^2$. This form happens to be exactly the form of NLSQ problems. For NLSQ we use lsqnonlin in Matlab using the trust-region-reflective algorithm [68].

4.2.2 LQR

Another approach is to treat the system as having linear dynamics and quadratic costs. When the dynamics are linear and the costs are quadratic there is a known optimal solution computed by the Linear-Quadratic Regulator (LQR). For a discrete system we can write the dynamics and costs as:

$$x_{i+1} = Ax_i + Bu_i \tag{4.22}$$

$$C(x_i, u_i) = \langle x_i, Qx_i \rangle + \langle u_i, Ru_i \rangle \tag{4.23}$$

$$C_{end}(x_N) = \langle x_N, Q_{end} x_N \rangle \tag{4.24}$$

where x_i is the i-th state, u_i is the i-th input, N is the number of steps in the trajectory, and Q, R, and Q_{end} are symmetric, positive definite matrices for the costs.

Then the formulation of the controller depends on whether the time horizon is finite or infinite. For the finite case we have:

$$J = \sum_{i=0}^{N-1} C(x_i, u_i) + C_{end}(x_N) \tag{4.25}$$

$$V(x_i) = \min_{u_i}(C(x_i, u_i) + V(Ax_i + Bu_i)) \tag{4.26}$$

where J is the total cost and V is the cost-to-go. Now the optimal control is to minimize u_i for every input so if we take the derivative with respect to u_i of the value function we can find the optimal input, u^*. Before doing that we know the V should also be represented as a quadratic form because it is the sum of quadratic forms so we get:

$$V(x_i) = \langle x_i, S_i x_i \rangle \tag{4.27}$$

where S_i is a symmetric, positive definite matrix. This allows for the rearrangement of the value equation to:

$$0 = \min_{u_i}(\langle x_i, Qx_i\rangle + \langle u_i, Ru_i\rangle + \langle Ax_i + Bu_i, S_{i+1}(Ax_i + Bu_i)\rangle) - \langle x_i, S_i x_i\rangle \tag{4.28}$$

Taking the derivative with respect to u we get:

$$0 = 2u^T R + 2B^T S_{i+1} A x_i + 2u^T B^T S_{i+1} B \tag{4.29}$$

solving for u will give the optimal input:

$$u^* = (R + B^T S_{i+1} B)^{-1}(B^T S_{i+1} A)x_i \tag{4.30}$$

So, we can determine the optimal input given the current state and the form of the next cost-to-go matrix. This means we need to determine S_i to compute the optimal input. To determine S_i we can plug the optimal input back into (4.28) and solve for S_i.

$$S_N = Q_{end} \tag{4.31}$$

$$S_i = A^T S_{i+1} A - (A^T S_{i+1} B)(R + B^T S_{i+1} B)^{-1}(B^T S_{i+1} A) + Q \tag{4.32}$$

To determine S_i we just start from the end point and compute backwards to the first step. The the optimal inputs can be computed. To proceed the first input is taken and the solution reoptimized again with a new approximation to the linear dynamics.

LQR is an important and useful control scheme because it is easy to implement and compute as long as A, B, Q, R, and Q_{end} are known. The values associated with costs are given, but for a dynamic system A and B may not be known or the dynamics are non-linear so they have to be numerically approximated. For a non-linear system we can use finite differences to approximate A and B about the starting position in the trajectory. If desired the optimization step can be iterated to

attempt to converge to better inputs for nonlinear systems. With $x_{i+1} = f(x_i, u_i)$ we approximate A and B as:

$$A \approx \frac{f(x_i + \epsilon, u_i) - x_{i+1}}{\epsilon} \tag{4.33}$$

$$B \approx \frac{f(x_i, u_i + \epsilon) - x_{i+1}}{\epsilon} \tag{4.34}$$

where ϵ is a small perturbation (e.g., 1×10^{-5}) and the vectors are perturbed element-wise.

Given: x_0, u, $f(x, u)$, N, Q, Q_{end}, R ;
A, B = linearize(f, x_0, u) ;
S_N = Q_{end};
for *i=N-1..1* **do**
  $S_i = A^T S_{i+1} A - (A^T S_{i+1} B)(R + B^T S_{i+1} B)^{-1}(B^T S_{i+1} A) + Q$;
end
return $u = -(R + B^T S_1 B)^{-1}(B^T S_1 A)x_0$;

Algorithm 3: LQR

4.2.3 iLQR

One weakness of LQR is that it only linearizes the dynamics about the initial point and so the linearization isn't necessarily accurate over the whole trajectory. To improve the accuracy we can linearize the dynamics at every point along the trajectory. The drawback here is due to A and B being state dependent we have to do several repetitions of LQR to converge to the optimal solution, hence iterated-LQR. The approach goes as: start with an initial input trajectory, determine the state trajectories and the linearized dynamics for the trajectory, compute the cost-to-go matrix backwards over the trajectory, update the inputs using the cost-to-go similar to LQR, and if this doesn't satisfy the convergence condition repeat.

The approach for iLQR is essentially the same as for LQR, the main difference is during the algorithm we need a forward and backward pass to optimize the trajectory. The forward pass consists of what the initial considered inputs are and collects the sequence of states and linearizes the dynamics at each step, the backward pass is then computing the cost-to-go, S_i, for each state

in the trajectory. With S_i computed the inputs can be updated so that they minimize the value as was done in LQR. Repeating the forward and backward pass converges onto the optimal input trajectory.

Given: x_0, $f(x, u)$, N, Q, Q_{end}, R ;
Initialize: u_i;
while *not converged* **do**
  for *i=0..N* **do**
    $x_{i+1} = f(x_i, u_i)$;
    A_i, B_i = linearize(f, x_i, u_i);
  end
  $S_N = Q_{end}$;
  for *i=N-1..0* **do**
    $S_i = A_i^T S_{i+1} A_i - (A_i^T S_{i+1} B_i)(R + B_i^T S_{i+1} B_i)^{-1}(B_i^T S_{i+1} A_i) + Q$;
    $u_i = -(R + B_i^T S_{i+1} B_i)^{-1}(B_i^T S_{i+1} A) x_i$;
  end
end
return u_0;

Algorithm 4: iLQR

4.3 Reinforcement Learning

For the MPC control the computation time is quite restricted by relying on the simulation and we'd like to avoid this. Machine learning offers a way to learn efficient to compute control policies and are much more plausible to run in real-time control applications. The difficulty in machine learning comes from properly training the system for the tasks that you want to accomplish.

The field of machine learning that focuses on learning control tasks is known as reinforcement learning [67]. Reinforcement learning (RL) in general works by exploring the environment and collecting samples of the states it reaches, the actions it takes, and the rewards it measures. Then uses the collected data to approximate the cost-to-go. Typically in RL the cost-to-go is referred to as the value and rewards are used instead of costs. Then, with the value function learned the control policy is able to maximize the value function at every step. There is some nuance based on the

problem being explored and the techniques for learning the value function, but we can generally separate approaches into model-based and model-free techniques.

A model-based reinforcement learning algorithm will in someway learn or approximate the dynamics of the system and use it for developing the policy. These approaches can overlap a bit with other machine learning fields, such as supervised learning. Model-based algorithms typically learn much quicker than model-free methods, but have a tendency to reach performance plateaus earlier and at lower performance thresholds.

Model-free methods learn a policy directly from the rewards it measures rather than using the dynamics of the system. These approaches have seen the best overall performance in complex control tasks, but need much more data to train. Model-free reinforcement learning algorithms have been the foundations for impressive feats such as playing video games [69] and learning the game go [70].

For our control tasks we want to focus on learning effective policies and prefer the training time to be as minimal as possible, which typically means using as few samples as possible, but we also need to consider the accuracy of the policy. We will explore several techniques for learning and see what the responses are.

Writing the general approach to reinforcement learning we can state the Bellman equation (4.15) in a more standard form:

$$V^{\pi}(x_t) = \sum_{i=t}^{\infty} \gamma^{t-i} R(x_t, \pi(x_t)) \tag{4.35}$$

where V^{π} is the value for a policy π, $\gamma \in [0, 1]$ is the discount factor, and R is the reward function. Which can be rearranged as:

$$V^{\pi}(x_t) = R(x_t, \pi(x_t)) + \gamma V^{\pi}(x_{t+1}) \tag{4.36}$$

which amounts to saying the optimal policy, π, optimizes the value function at every step. So, by collecting many samples of the state, actions, and rewards the value can be approximated and the policy derived. However, it is more typical to use an augmented value, the Q-function [67].

$$Q^{\pi}(s_t, a_t) = R(x_t, a_t) + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) \tag{4.37}$$

where the value is explicitly dependent on the action, a_t.

Learning the value and policy are then dependent on the algorithmic approach and the system being studied.

4.3.1 Deep Deterministic Policy Gradient

One powerful approach to reinforcement learning the the Deep Deterministic Policy Gradient (DDPG) [71]. This is a model-free method designed to work with dynamic systems that have continuous state spaces and continuous action spaces. The idea behind DDPG is to approximate both Q and π with deep neural networks and to train Q to minimize the error in the value and the policy is trained to maximize the value. Some modifications to the base algorithm have to be included to ensure the learning process is stable.

The error for training Q is defined as:

$$L = \frac{1}{N}\sum_{i}(R(x_i, a_i) + \gamma Q(x_{i+1}, \pi(x_{i+1}|\theta_{\pi})|\theta_Q) - Q((x_i, a_i)|\theta_Q))^2 \tag{4.38}$$

where N is the number of sampled points, i indicates a sample of states, θ_Q are the parameters for Q, and θ_{π} are the parameters for the policy. Taking the gradient with respect to θ_Q gives the update rule for the Q network.

For the policy, the policy gradient used for the updates is defined as [72]:

$$\nabla_{\theta_{\pi}} J = \frac{1}{N}\sum_{i} \nabla_a Q(x_i, a_i|\theta_q)|_{a_i=\pi(x_i)} \nabla_{\theta_{\pi}} \pi(s_i|\theta_{\pi}) \tag{4.39}$$

where J is the expected return.

Using these updates enables the simultaneous training of both the Q and π networks. However, DDPG uses two techniques to get stable learning: a replay buffer and target networks. The replay buffer stores past observations and then samples minibatches to update the networks to keep all

samples relevant. Target networks are originally copies of the networks that gradually follow the updates of the networks to smooth the learning process as minor changes to the value can result in large policy changes. The implementation uses Keras-rl [73].

Initialize θ_Q and θ_π for Q and π and the target networks Q' and π';
Initialize the replay buffer R;
for *episode=1...M* **do**
  start in initial state x_1;
  for *t=1...T* **do**
    $a_t = \pi(x_t)$;
    $x_{t+1} = f(x_t, a_t)$;
    add (x_t, a_t, r_t, x_{t+1}) to R;
    sample N minibatches from R;
    update θ_Q by minimizing: $\frac{1}{N}\sum_i (r_i + \gamma Q'(x_{i+1}, \pi'(x_{i+1}|\theta'_\pi)|\theta'_Q) - Q(x_i, a_i)|\theta_Q)^2$;
    update θ_π using: $\nabla_{\theta_\pi} J = \frac{1}{N}\sum_i \nabla_a Q(x_i, a_i|\theta_q)|_{a_i=\pi(x_i)} \nabla_{\theta_\pi}\pi(s_i|\theta_\pi)$;
    update target networks: $\theta'_Q \leftarrow \tau\theta_Q + (1-\tau)\theta'_Q$, $\theta'_\pi \leftarrow \tau\theta_\pi + (1-\tau)\theta'_\pi$;
  end
end

Algorithm 5: DDPG algorithm

4.3.2 Guided Policy Search

A hybrid of model-based and model-free methods that accelerates the learning of policies is the Guided Policy Search (GPS) [45]. GPS develops a local linear Gaussian models to the dynamics rather than global dynamics models and uses a linear Gaussian controller for trajectory generation. The learned policy is then trained in a supervised fashion to match the controller generated trajectories.

GPS uses an algorithm similar to iLQR to generate its trajectories, iterative Linear Quadratic Gaussian Regulator (iLQG). This algorithm works similarly to iLQR except there is now Gaussian noise included into the system. However, GPS also operates under the consideration that the dynamics are unknown (they can be sampled, but no analytic function). So, in order to come up with the control policies the algorithm needs to come up with a model of the dynamics, optimize the trajectory using iLQG, and train the learned policy on the generated policies.

The procedure for GPS starts by initially collecting samples of the dynamics from randomly given input trajectories. The algorithm then repeats the steps of generating new trajectories for the linear-Gaussian controllers, fitting local Gaussian models of the dynamics to the collected sample trajectories, training the policy, and incrementing some internal update parameters. The model of the dynamics has two components. The local dynamics model is a linear Gaussian model while a Gaussian mixture model (GMM) is used globally to estimate priors for the iLQG algorithm. For training the policy the objective is to minimize the KL-Divergence between the linear-Gaussian policies and the trained policy. The reason for minimizing the KL-Divergence is to make sure the change in policies between steps is smoothed to get more stable policies. There are several details for the algorithm and optimization that are discussed in [45], but the general idea is to use iLQG to generate policies from Gaussian models of the sampled dynamics and train a neural-network policy to mimic them in a supervised way. The implementation used the Robotic Learning Lab's implementation [74].

for *i=1..N* **do**
    using controllers p_i generate new trajectories ;
    fit dynamics to collected trajectories ;
    minimize KL-Divergence to update the learned policy π ;
    update internal variables;
end

Algorithm 6: GPS algorithm

4.4 Comparisons

To compare the different control schemes tested we will first compare the optimization based methods and then the reinforcement learning methods. The performance will be compared across several different test problems. Through this we aim to find what are the utility of the different control approaches, simulations, and what works and what does not.

It should be noted that the explicit Cosserat simulation turns out to not be particularly useful for optimization or learning while the other schemes have their benefits and drawbacks for the different

control and learning tasks. The difficulty with the explicit Cosserat simulation comes primarily from the shortness of the time steps and vibrations in the axial deformations. Due to having very short time steps the state between time steps changes very little which makes approximating the dynamics more difficult mostly because it acts like an identity map locally, one way to deal with this is to run consecutive steps and call them a single step, but this leads to higher computational demand which we are trying to avoid. The problem with axial vibrations is that they are much more rapid than the bending or torsion vibrations. The issue with this is the primary deformations are bending which are qualitatively what should be controlled, but the presence of axial vibrations effectively causes lots of noise in the deformation process as well as delaying the bending response until the vibrations have damped out. Interpreting these vibrations as noise makes it much harder to predict the dynamics and causes the control and learning to either be expensive or inconsistent. Effectively the relevant time scale is much larger than the time step which makes the optimization too computationally demanding to be useful. This leads to only comparing the performance of the implicit scheme and the explicit Kirchhoff scheme.

4.4.1 MPC Comparisons

For MPC we used three approaches: NLSQ, LQR, and iLQR. It should be noted that the implicit scheme works well for NLSQ as the control can be defined in the task space, but is very inefficient with LQR and iLQR as those schemes need to approximate the jacobians in the state space which is a very expensive operation to do on the implicit scheme. This makes the Kirchhoff scheme the only practical model for use with LQR and iLQR. It is also necessary to point out that at short time horizons the Kirchhoff scheme aren't very effective, so the experiments start at a 5 step horizon so the performance is decent for all trials.

Dynamic Reaching

The dynamic reaching task consisted of randomly selecting a point in the dynamic workspace and controlling the simulation to get the tip as close to that point as possible. To be consistent across comparisons the same 10 points were used in testing which can be seen in Figure 4.3.

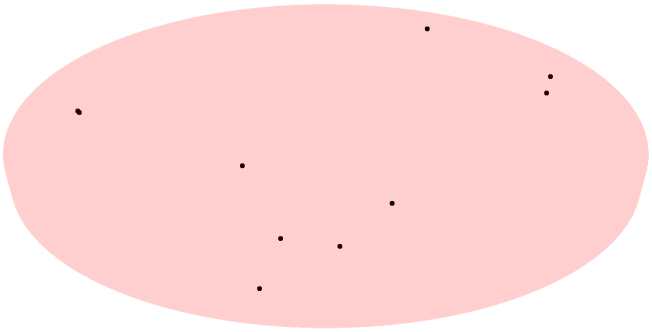
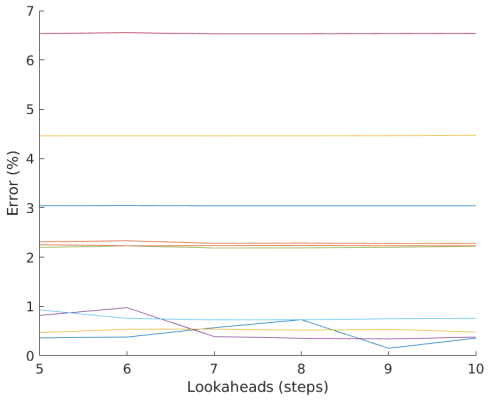


Figure 4.3: The randomly selected points for the dynamic reaching task for the optimization control schemes.

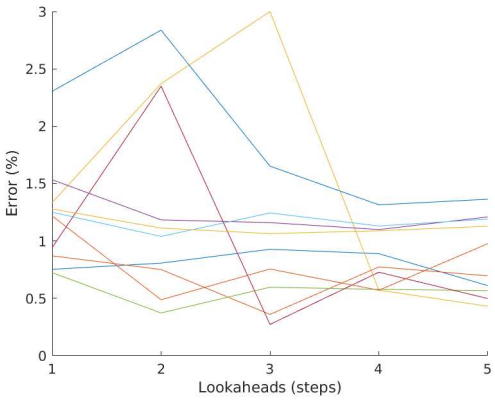
For the experiments on the dynamic reaching task we terminate the experiment if it has reached an error of less than 5% or after running for 100 total steps and we record the lowest error achieved. Since MPC can look ahead a variable number of steps and potentially improve the trajectory we want to see how the accuracy compares with number of lookaheads. The trends are seen in Figure 4.4.

It can be seen that nearly every trial gets below the 5% threshold which means the trends are not too interesting to look at as the solutions are considered sufficient in all cases. For the few that did not reach the threshold, increasing the steps did not seem to have any influence on the performance. It can be noted that due to the large time steps in the implicit scheme it is possible that the experiment skips over the target point in discrete time when it would have passed through it in continuous time which could influence the error a bit.

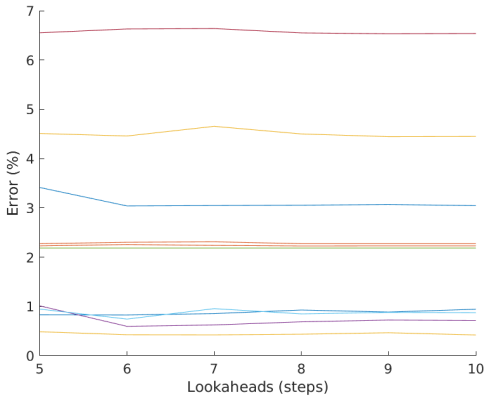
To see how efficiently the control algorithms run we want to see what the real-time ratios are and how they are influenced by the scheme and number of lookaheads. For the dynamic reaching task, Figure 4.5, we see that implicit schemes with larger time steps are less sensitive to the number of lookaheads and having a larger time horizon makes the optimization slower. We note that only



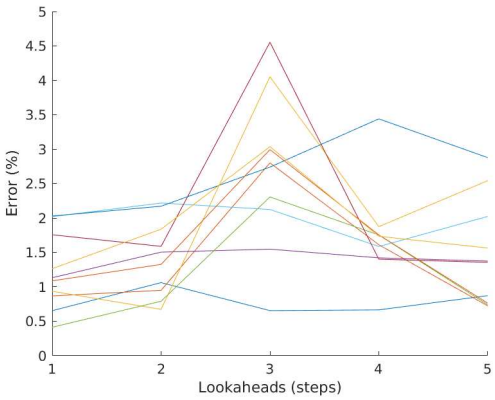
(a) NLSQ Kirchhoff.



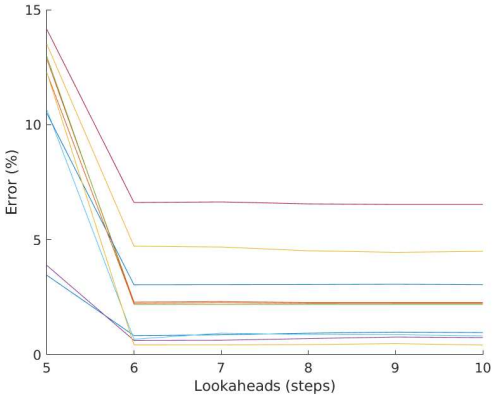
(b) NLSQ Implicit, $\Delta t = 0.01$.



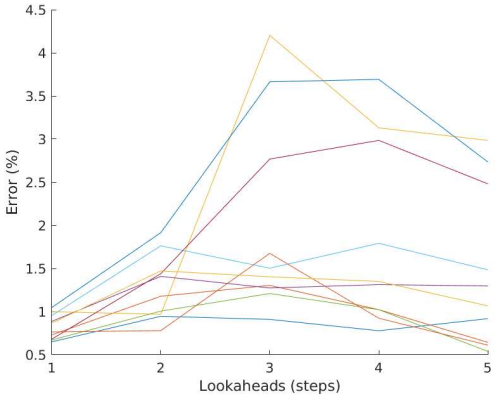
(c) LQR Kirchhoff.



(d) NLSQ Implicit, $\Delta t = 0.05$.



(e) iLQR Kirchhoff.



(f) NLSQ Implicit, $\Delta t = 0.1$.

Figure 4.4: The error in the tip position for the dynamic reaching task with the different control and simulations. The error is plotted against the number of lookaheads used in the optimization.

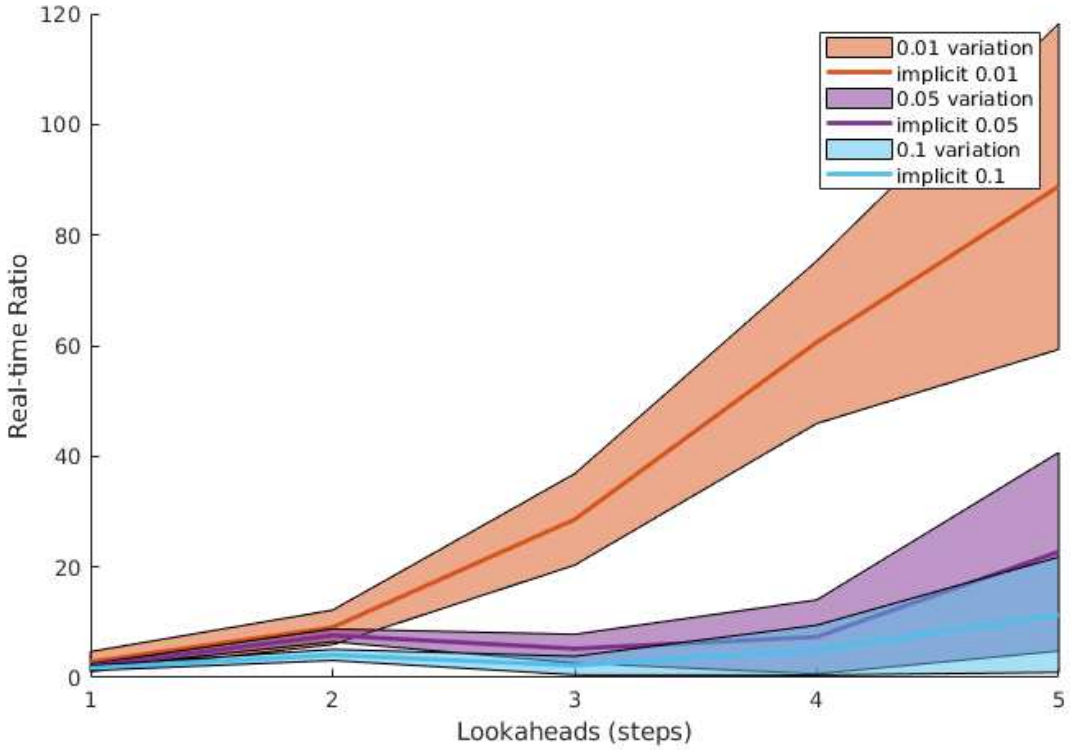


Figure 4.5: Shows the mean and standard deviation for the real-time ratios for the NLSQ algorithm using the implicit simulation scheme on the dynamic reaching task. The other algorithms and schemes are not displayed due to having significantly larger ratios.

the results for the NLSQ algorithm using the implicit simulation are shown. This is due to the other algorithms being significantly slower and not worth showing. We see that the only setups that can achieve a practical real-time ratio are the implicit schemes with larger time steps.

Static Reach and Hold

For the static reach and hold task, a sample of points in the static workspace are selected and the control schemes attempt to reach the point and keep the tip near it. Again, the same 10 points are used for each trial and can be seen in Figure 4.6. For this task the error is also the tip error, but the termination condition is to also reach a small tip error and to have the last few time steps to have little variation in their error as to consider it a hold. The measured error is the average over the final few time steps.

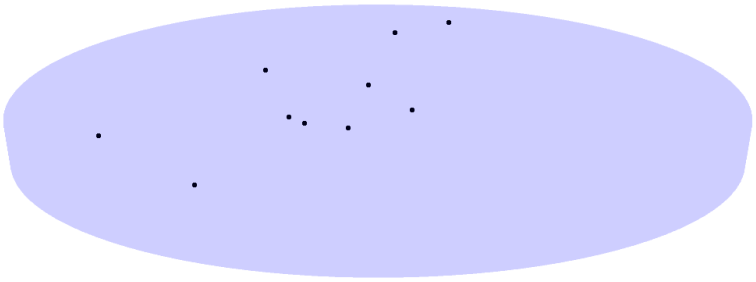
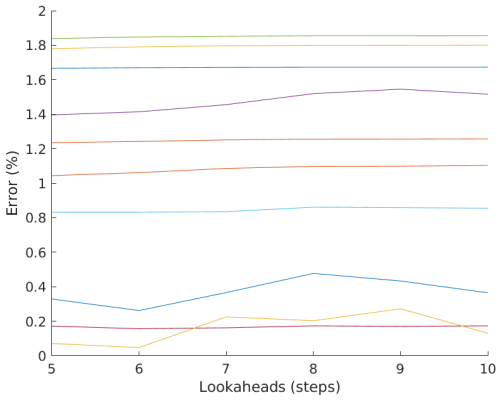
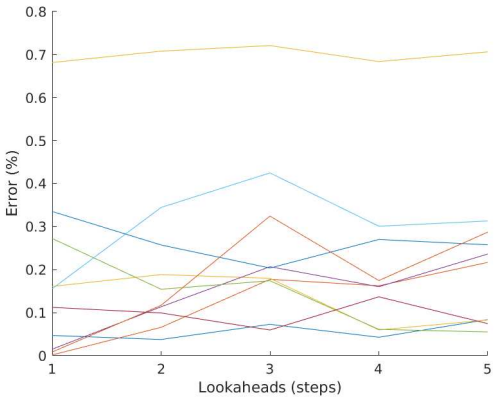


Figure 4.6: The randomly selected points for the static reach and hold task for the optimization control schemes.

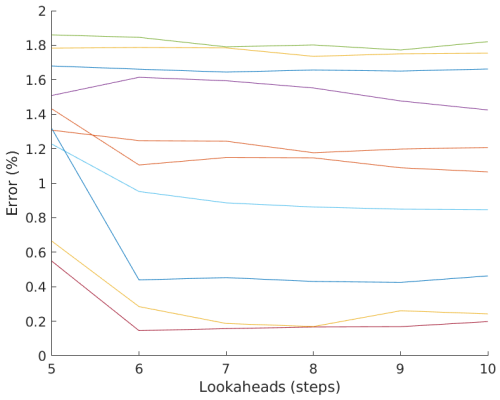
For the static reaching we see in general the control schemes were all quite successful and that increasing the number of lookahead points likely has little influence due to the termination threshold always being reached.



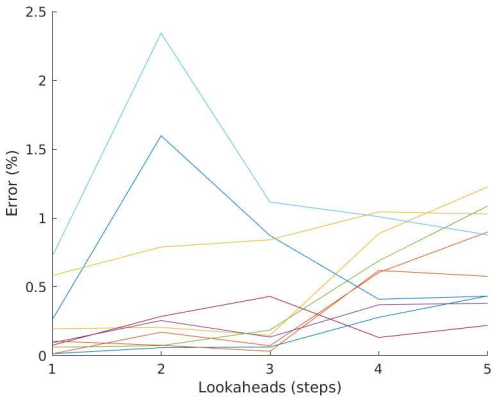
(a) NLSQ Kirchhoff.



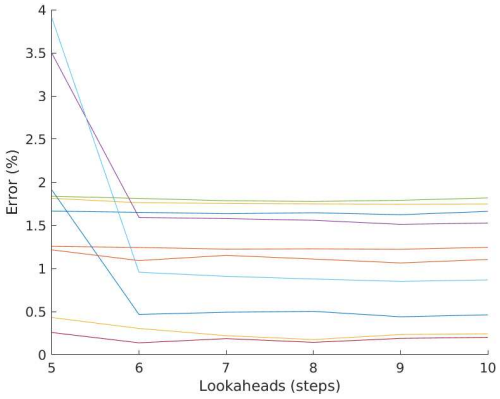
(b) NLSQ Implicit, $\Delta t = 0.01$.



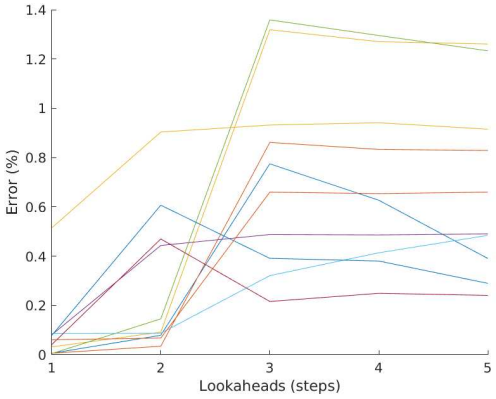
(c) LQR Kirchhoff.



(d) NLSQ Implicit, $\Delta t = 0.05$.



(e) iLQR Kirchhoff.



(f) NLSQ Implicit, $\Delta t = 0.1$.

Figure 4.7: The error in the tip position for the static reaching task with the different control and simulations. The error is plotted against the number of lookaheads used in the optimization.

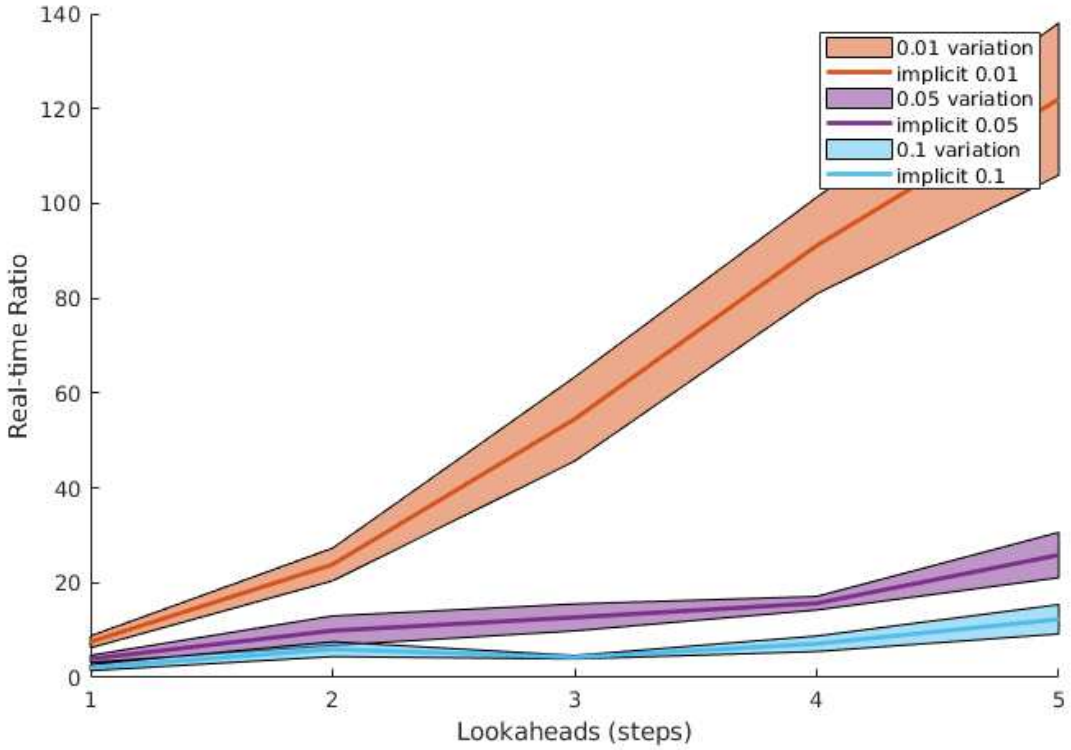


Figure 4.8: Shows the mean and standard deviation for the real-time ratios for the NLSQ algorithm using the implicit simulation scheme on the static reaching task. The other algorithms and schemes are not displayed due to having significantly larger ratios.

For static reaching real-time ratios, Figure 4.8, we see the same trend as in the dynamic reaching. Longer time horizons increase the computation time and larger time steps make it easier to achieve low real-time ratios.

Trajectory Tracking

For the trajectory tracking we select circles similar to the quasi-static control scheme such that the circles are within the static workspace. We record the average error in the tracking experiment and see the influence of the time horizon on the results.

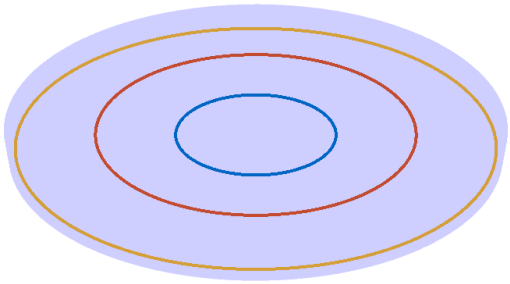
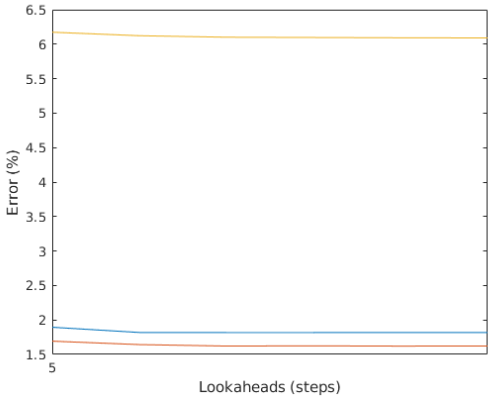


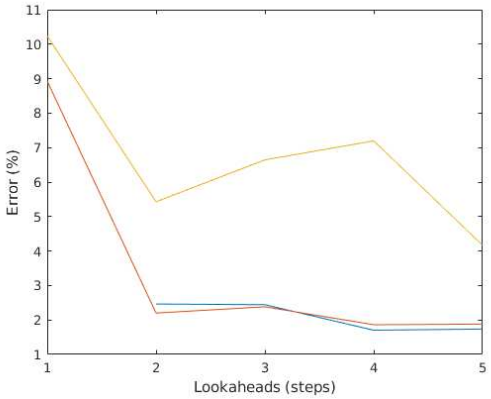
Figure 4.9: The traces of the trajectories to be tracked by the manipulators. The paths are kept within the static workspace to gaurantee that the points are always reachable.

We can see in general it was harder to track the larger circles, likely due to requiring higher speeds to stay with the trajectory. It is also noteworthy that the large time step implicit scheme did slightly worse in the tracking, most likely due to the low time resolution of the command inputs. Here we also see that more lookaheads may not necessarily be better, but there is likely an optimum for a given situation.

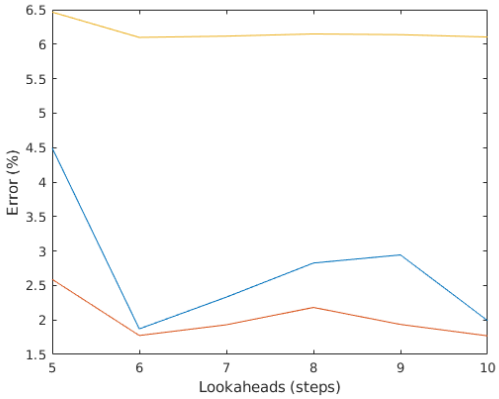
The trajectory tracking shows the same trend in the real-time ratios as the reaching tasks.



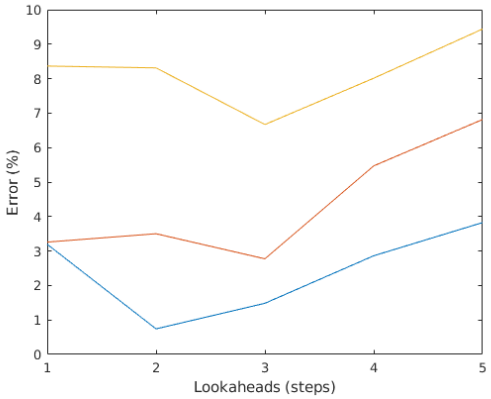
(a) NLSQ Kirchhoff.



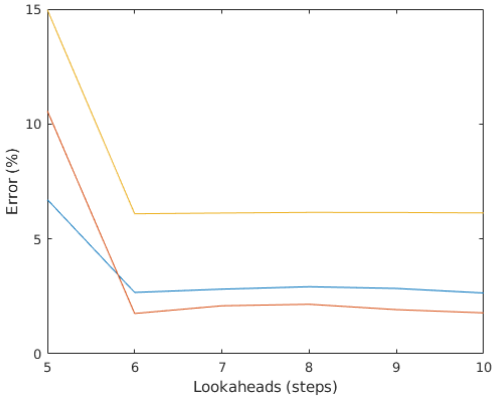
(b) NLSQ Implicit, $\Delta t = 0.01$.



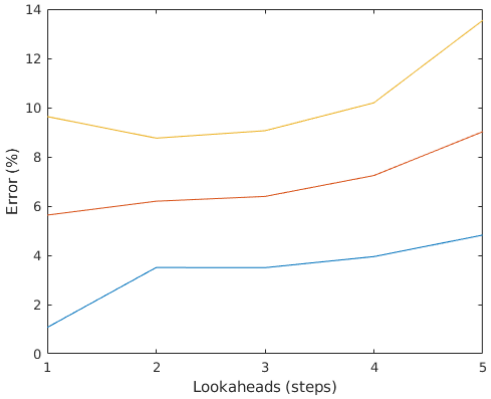
(c) LQR Kirchhoff.



(d) NLSQ Implicit, $\Delta t = 0.05$.



(e) iLQR Kirchhoff.



(f) NLSQ Implicit, $\Delta t = 0.1$.

Figure 4.10: The average error in the tip position for the trajectory tracking task with the different control and simulations. The error is plotted against the number of lookaheads used in the optimization. Blue is the inner circle, red is middle circle, and yellow is outer circle.

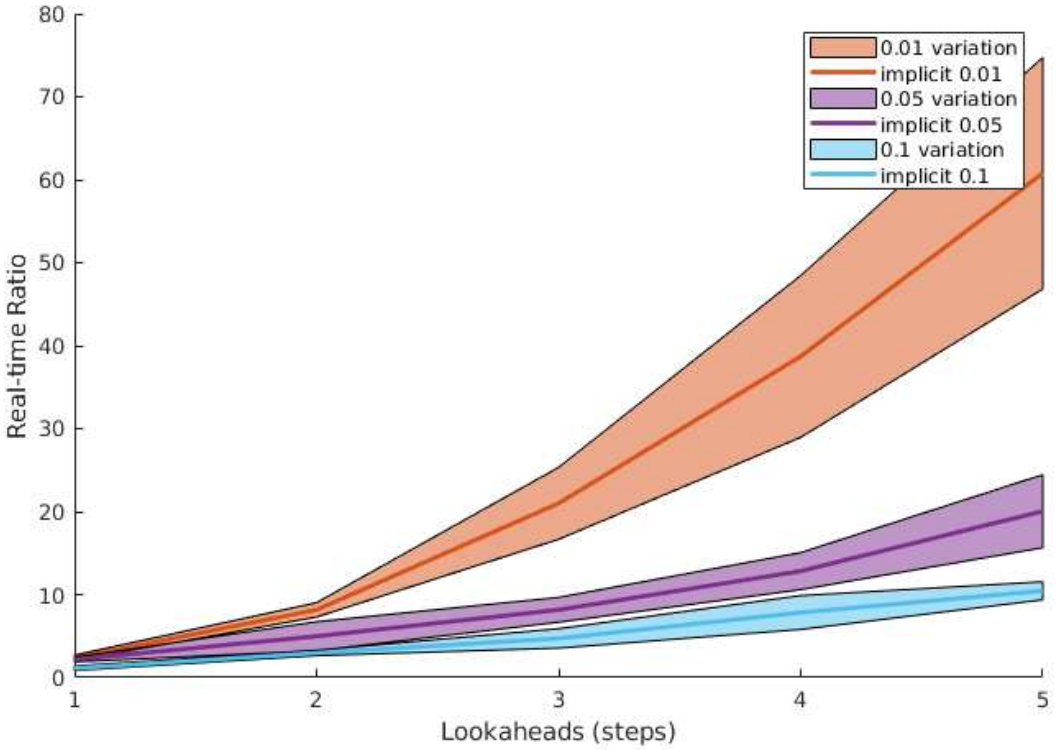


Figure 4.11: Shows the mean and standard deviation for the real-time ratios for the NLSQ algorithm using the implicit simulation scheme on the trajectory tracking task. The other algorithms and schemes are not displayed due to having significantly larger ratios.

Overall

For all the experiments the only setup that was able to achieve a real-time ratio less than 1 was the NLSQ control with the implicit scheme using a time step of $\Delta t = 0.1$ and 1 step of lookahead. While it is possible to improve the running times and algorithms it is unlikely that the time step will come down significantly with more desirable parameters. So, we can say it is possible to achieve real-time control through optimization, but it is not practical in these demonstrations. In the results we can see, in the Kirchhoff case, that once past a certain time horizon the performance doesn't seem to improve.

The conclusions we can draw for the optimization based control is that it is quite difficult to achieve real-time control for the soft manipulators especially as they get more complicated. It is possible that the efficiency of the optimization algorithms can be improved, but it is unlikely that it will improve the results to a practical level. It is seen that the implicit scheme is certainly beneficial for faster control schemes as the longer time steps are a significant benefit; however, too long of a time step decreases the resolution in the control command frequency which is undesirable. We can also infer that the task space is a much better space to define the objective in rather than the state space. The LQR based optimizations used state space costs which resulted in significant slow-down due to linearizing the dynamics while NLSQ was only posed in the task space which appears to have helped the computation time without losing accuracy and effectiveness. Unfortunately, it appears that optimization based control is a bit out of reach for practical implementations, but the most successful approach likely would need an implicit simulation and task space objectives.

4.4.2 Learning Comparisons

For the learning based control we have a few different objectives for the testing. In general we know that the resulting control policies should run in real-time, so that is not an important measure here. However, the learning algorithms require a certain number of samples of the soft manipulators behavior to be effective and for the sake of practicality fewer necessary samples is desired. Also, reinforcement learning algorithms tend to have a single set task whereas robots

typically have a more general task (e.g., reinforcement learning trains to reach a specific point while a manipulator needs to reach arbitrary points). We want to test how well the learning generalizes these tasks when including the goal or target into the state measurements. We also want to test how well they generalize to more difficult or complex tasks. So, for the learning we use a slightly different set of control tasks to measure the performance.

The first task is reaching a single point that is in the dynamic workspace and not the static workspace. For this the policy is trained to only reach a single point so that it is more akin to typical RL problems. Since this is reaching a point in the dynamic workspace the policy cannot hold that position, so we need to set a terminating condition to say that reaching the point was successful and the training can be reset. For this we consider being within 5% of the desired point relative to the length of the body to have successfully achieved the task.

The next task generalizes the first task, be able to reach any point within the dynamic workspace. Now, this problem allows any point in the dynamic workspace to be selected not just a dynamically reachable point as before. We also want a single policy to be able to generalize to any of these points. So for this we randomly sample points in the dynamic workspace for the policy to try to reach and since some points are only dynamically reachable we use the same terminating condition as before.

The final task is to track a trajectory. Since we want an arbitrary trajectory to be tracked we train by generating random trajectories through the static workspace. To generate a random trajectory we pick a random starting and ending point and generate small random initial and final velocities then a fourth-order polynomial is fit to those values. In this case we want the manipulator to track the trajectory by being both close to the desired point and the desired velocity. To test the performance of training we have the manipulator track a circular trajectory like in the MPC control.

For all tasks the reward is defined as:

$$R = \begin{cases} -\frac{\|p(L)-p_{des}\|}{\beta L}, & \|p(L) - p_{des}\| \leq \beta L \\ -1, & \text{otherwise} \end{cases} \tag{4.40}$$

where $\beta \in (0, 1]$ and p_{des} is the desired tip position. This reward essentially says that once the manipulator is within a certain distance to the desired position give a better reward the closer it is to the target, but if outside this range just give a negative reward. The reason for this range is that it seems DDPG works better with rewards in this range [75]. We select $\beta = 0.3$ as that seems to give good policies.

Each of these tasks will be trained on using DDPG and GPS and their performance will be discussed in the following.

4.4.3 DDPG Performance

For DDPG we use more samples on the different tasks to account for the complexity. Also, we use the implicit scheme for generating the sample data, the change in the states between steps in the explicit schemes seem to make the performance worse. For the dynamic reaching task used a total of 500000 samples for each training and out of the 10 points tested all 10 were reached. The 500000 samples seem to generally be excessive. For the arbitrary target task 500000 samples points were used again. To test the performance the policy was given 50 random points to reach and the total reached and the best tip error were recorded. For 5 trials of training the policy the average number of points successfully reached was 83.6% and have an average tip error of 5.2%. For the trajectory tracking the policy was trained on 1000000 samples and the learned policy had an average tracking error for the circle of 6.4%.

Overall it appears that the DDPG algorithm did reasonably well on the given control tasks. More testing could have been done on the number of samples used, but these are generally a lot of samples and we would prefer having to collect less data to successfully train the policies.

GPS Performance

In the GPS training the implicit scheme was used as well and more training was allowed for the more complex tasks. For the dynamic reaching task GPS was able to consistently reach the desired target after 5000 samples for all 10 out of the 10 points tested.

For the variable target task the average minimum tip error went down to roughly 2.5% and successfully reached the desired point 94.0% of the time after 250000 samples averaged across 10 trials.

For arbitrary trajectory tracking GPS achieved a 10.3% average tracking error for the circle after 500000 samples.

Overall

Both algorithms could be said to have achieved desirable control performance for the given tasks. However, GPS appeared to be a bit better in performance on the simpler tasks, DDPG better on the trajectory tracking, and GPS significantly better in the number of training samples. This makes it seem that GPS is the better option in these particular cases. It is possible that for more complex tasks or robots a model-free approach has the advantage as it generally gets better long-term performance.

There are several extensions to the methods that can be used to potentially get better results. We have attempted to use the Temporal Difference Models [47] and Hindsight Experience Replay [48] which claim to improve model-free techniques; however, we found the performance and number of samples to be nearly the same as DDPG and didn't have any benefit.

4.4.4 Control Comparisons

Both the optimization and learning based control can derive acceptable controllers for the simulated soft manipulator; however, given the computation time the learning based approaches seem to have the advantage. The only optimization control schemes that could achieve real-time performance only had one-step of lookahead and fairly low sampling rates. The performance of the algorithms could improve, but it seems unlikely that efficiency would improve enough to get the desired real-time performance. Since machine learning seems to be the better approach it is desired to make that as effective as possible, which typically means having the training be as efficient as possible to be able to prototype and control the robots as efficiently as possible. Given the millions of samples it takes for model-free algorithms to work model-based approaches become more at-

tractive even if the performance is possibly lower. Though in our testing the performance seemed to be similar between the methods. The primary downside to learning is the need to retrain for every prototype, but lower samples mitigates that issue somewhat.

Table 4.1: The measured performance of the different control schemes. The two important factors included are the real-time ratio and the accuracy. Here the real-time ratio is only reported for the cases that could achieve a value less than 1. For the reaching tasks the successful out of the attempted are reported for the MPC algorithms and the percentage for the RL algorithms (success is less than 5% error). For the trajectory tracking the average middle circle value is reported.

	NLSQ Implicit			NLSQ Kirchhoff	LQR	iLQR	DDPG	GPS
$\Delta t =$	0.01	0.05	0.1				0.05	0.05
Real-time	No	No	0.95	No	No	No	0.01	0.01
Dynamic Reach (single point)	NA	NA	NA	NA	NA	NA	100%	100%
Dynamic Reach (any point)	10/10	10/10	10/10	9/10	9/10	9/10	83.6%	94.0%
Static Reach	10/10	10/10	10/10	10/10	10/10	10/10	NA	NA
Trajectory	3.45%	4.36%	6.90%	1.64%	2.03%	3.38%	6.40%	10.3%

4.5 Summary

In this chapter we discussed the control of the soft manipulators and how effective the schemes were as well as the real-time capabilities. We began with a quasi-static controller that works, but is unlikely to be useful in a real control situation. Then, we moved on to Model Predictive Control and Reinforcement Learning. Here we saw that it is difficult to achieve real-time control using Model Predictive Control techniques with only the Nonlinear Least-Squares optimization with the implicit scheme using one fairly long lookahead being able to achieve real-time control. For Reinforcement Learning it was much more successful at achieving real-time control at the expense of training time. For the algorithms compared DDPG did better on the more complex task while GPS took significantly fewer samples, both performed well on the simpler tasks.

Chapter 5

Conclusions and Future Work

In this thesis we demonstrate the development of a Cosserat rod based modeling framework, the implementations of the simulations, and possible approaches to control for these soft robots. In the modeling we are able to capture the significant modes of deformations for the slender manipulators, which can be extended to higher dimensions later, and developed models for the common types of actuators: cables, fluidic, and artificial muscles. For the simulations we are able to implement both explicit and implicit integrators that preserve the group structure of the kinematics and eliminate numerical damping, often neglected aspects of the simulations, that can run in real-time, in practice we see that the CFL condition is quite restrictive for the explicit scheme and it appears the implicit scheme seems to be the preferred simulation scheme. For the control, we demonstrate the difficulty of getting real-time control with optimization based approaches and as a comparison show that Reinforcement Learning is an excellent technique for getting real-time control policies at the expense of training time. In the end, we see that real-time simulations and control are possible to develop for soft manipulators.

The work shown here is intended to be used as a basis for continuing to develop understanding of soft robots as well as analyzing more complex structures the combine both rigid and soft components. There are still many potential avenues to explore. For the modeling we can generalize to higher dimensional bodies than rods and can mix rigid and soft components. For the simulations, we have room to generalize the integration schemes to higher-order to get better accuracy, more minor optimizations, and generalizing to more complex material properties and geometric structures. Then the control, we can explore how to improve machine learning techniques to be more sample efficient and have better performance. There is also the avenue of exploring Lyanpunov stability derived controllers so that analytic real-time controllers can be developed.

Bibliography

[1] Deepak Trivedi, Christopher D Rahn, William M Kier, and Ian D Walker. Soft robotics: Biological inspiration, state of the art, and future research. *Applied Bionics and Biomechanics*, 5(3):99–117, 2008.

[2] Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521(7553):467–475, 2015.

[3] Sangbae Kim, Cecilia Laschi, and Barry Trimmer. Soft robotics: a bioinspired evolution in robotics. *Trends in biotechnology*, 31(5):287–294, 2013.

[4] Cecilia Laschi and Matteo Cianchetti. Soft robotics: new perspectives for robot bodyware and control. *Frontiers in bioengineering and biotechnology*, 2, 2014.

[5] G Dogangil, BL Davies, and F Rodriguez y Baena. A review of medical robotics for minimally invasive soft tissue surgery. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, 224(5):653–679, 2010.

[6] Paxton Maeder-York, Tyler Clites, Emily Boggs, Ryan Neff, Panagiotis Polygerinos, Dónal Holland, Leia Stirling, Kevin Galloway, Catherine Wee, and Conor Walsh. Biologically inspired soft robot for thumb rehabilitation. *Journal of Medical Devices*, 8(2):020933, 2014.

[7] Yahya Elsayed, Augusto Vincensi, Constantina Lekakou, Tao Geng, CM Saaj, Tommaso Ranzani, Matteo Cianchetti, and Arianna Menciassi. Finite element analysis and design optimization of a pneumatically actuating silicone module for robotic surgery applications. *Soft Robotics*, 1(4):255–262, 2014.

[8] Panagiotis Polygerinos, Zheng Wang, Kevin C Galloway, Robert J Wood, and Conor J Walsh. Soft robotic glove for combined assistance and at-home rehabilitation. *Robotics and Autonomous Systems*, 73:135–143, 2015.

[9] J. Burgner-Kahrs, D. C. Rucker, and H. Choset. Continuum robots for medical applications: A survey. *IEEE Transactions on Robotics*, 31(6):1261–1280, Dec 2015.

[10] Andrew D Marchese, Cagdas D Onal, and Daniela Rus. Autonomous soft robotic fish capable of escape maneuvers using fluidic elastomer actuators. *Soft Robotics*, 1(1):75–87, 2014.

[11] M. Giorelli, F. Renda, G. Ferri, and C. Laschi. A feed-forward neural network learning the inverse kinetics of a soft cable-driven manipulator moving in three-dimensional space. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5033–5039, Nov 2013.

[12] D. Trivedi, A. Lotfi, and C. D. Rahn. Geometrically exact models for soft robotic manipulators. *IEEE Transactions on Robotics*, 24(4):773–780, Aug 2008.

[13] M. Giorelli, F. Renda, M. Calisti, A. Arienti, G. Ferri, and C. Laschi. A two dimensional inverse kinetics model of a cable driven manipulator inspired by the octopus arm. In *2012 IEEE International Conference on Robotics and Automation*, pages 3819–3824, May 2012.

[14] B. A. Jones and I. D. Walker. Kinematics for multisection continuum robots. *IEEE Transactions on Robotics*, 22(1):43–55, Feb 2006.

[15] Srinivas Neppalli, Matthew A. Csencsits, Bryan A. Jones, and Ian D. Walker. Closed-form inverse kinematics for continuum manipulators. *Advanced Robotics*, 23(15):2077–2091, 2009.

[16] Manuel Schaffner, Jakob A Faber, Lucas Pianegonda, Patrick A Rühs, Fergal Coulter, and André R Studart. 3d printing of robotic soft actuators with programmable bioinspired architectures. *Nature communications*, 9(1):878, 2018.

[17] Ming Luo, Mahdi Agheli, and Cagdas D Onal. Theoretical modeling and experimental analysis of a pressure-operated soft robotic snake. *Soft Robotics*, 1(2):136–146, 2014.

[18] Norihiro Kamamichi, Masaki Yamakita, Kinji Asaka, and Zhi-Wei Luo. A snake-like swimming robot using ipmc actuator/sensor. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1812–1817. IEEE, 2006.

[19] Huai-Ti Lin, Gary G Leisk, and Barry Trimmer. Goqbot: a caterpillar-inspired soft-bodied rolling robot. *Bioinspiration & biomimetics*, 6(2):026007, 2011.

[20] Takuya Umedachi, Vishesh Vikas, and Barry A Trimmer. Highly deformable 3-d printed soft robot generating inching and crawling locomotions with variable friction legs. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4590–4595. IEEE, 2013.

[21] S. Seok, C. D. Onal, K. J. Cho, R. J. Wood, D. Rus, and S. Kim. Meshworm: A peristaltic soft robot with antagonistic nickel titanium coil actuators. *IEEE/ASME Transactions on Mechatronics*, 18(5):1485–1497, Oct 2013.

[22] Sung-Weon Yeom and Il-Kwon Oh. A biomimetic jellyfish robot based on ionic polymer metal composite actuators. *Smart materials and structures*, 18(8):085002, 2009.

[23] Kwangmok Jung, Ja Choon Koo, Young Kwan Lee, Hyouk Ryeol Choi, et al. Artificial annelid robot driven by soft actuators. *Bioinspiration & biomimetics*, 2(2):S42, 2007.

[24] Michael T Tolley, Robert F Shepherd, Bobak Mosadegh, Kevin C Galloway, Michael Wehner, Michael Karpelson, Robert J Wood, and George M Whitesides. A resilient, untethered soft robot. *Soft robotics*, 1(3):213–223, 2014.

[25] Federico Renda, Michele Giorelli, Marcello Calisti, Matteo Cianchetti, and Cecilia Laschi. Dynamic model of a multibending soft robot arm driven by cables. *IEEE Transactions on Robotics*, 30(5):1109–1122, 2014.

[26] E. Acome, S. K. Mitchell, T. G. Morrissey, M. B. Emmett, C. Benjamin, M. King, M. Radakovitz, and C. Keplinger. Hydraulically amplified self-healing electrostatic actuators with muscle-like performance. *Science*, 359(6371):61–65, 2018.

[27] Bianca S Homberg, Robert K Katzschmann, Mehmet R Dogar, and Daniela Rus. Haptic identification of objects using a modular soft robotic gripper. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1698–1705. IEEE, 2015.

[28] Jiefeng Sun, Benjamin Pawlowski, and Jianguo Zhao. Embedded and controllable shape morphing with twisted-and-coiled actuators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[29] Oliver M O'Reilly. *Modeling nonlinear problems in the mechanics of strings and rods*. Springer, 2017.

[30] Thomas George Thuruthel, Yasmin Ansari, Egidio Falotico, and Cecilia Laschi. Control strategies for soft robotic manipulators: A survey. *Soft robotics*, 5(2):149–163, 2018.

[31] D. C. Rucker and R. J. Webster III. Statics and dynamics of continuum robots with general tendon routing and external loading. *IEEE Transactions on Robotics*, 27(6):1033–1044, Dec 2011.

[32] J. Shintake, S. Rosset, B. E. Schubert, D. Floreano, and H. R. Shea. A foldable antagonistic actuator. *IEEE/ASME Transactions on Mechatronics*, 20(5):1997–2008, Oct 2015.

[33] Mohsen Shahinpoor, Yoseph Bar-Cohen, JO Simpson, and J Smith. Ionic polymer-metal composites (ipmcs) as biomimetic sensors, actuators and artificial muscles-a review. *Smart materials and structures*, 7(6):R15, 1998.

[34] Carter S Haines, Márcio D Lima, Na Li, Geoffrey M Spinks, Javad Foroughi, John DW Madden, Shi Hyeong Kim, Shaoli Fang, Mônica Jung de Andrade, Fatma Göktepe, et al. Artificial muscles from fishing line and sewing thread. *science*, 343(6173):868–872, 2014.

[35] III Robert J. Webster and Bryan A. Jones. Design and kinematic modeling of constant curvature continuum robots: A review. *The International Journal of Robotics Research*, 29(13):1661–1683, 2010.

[36] Federico Renda, Frédéric Boyer, Jorge Dias, and Lakmal Seneviratne. Discrete cosserat approach for multisection soft manipulator dynamics. *IEEE Transactions on Robotics*, 34(6):1518–1533, 2018.

[37] Raymond H Plaut. Mathematical model of inchworm locomotion. *International Journal of Non-Linear Mechanics*, 76:56–63, 2015.

[38] Yoel Shapiro, Kosa Gabor, and Alon Wolf. Modeling a hyperflexible planar bending actuator as an inextensible euler–bernoulli beam for use in flexible robots. *Soft Robotics*, 2(2):71–79, 2015.

[39] Javad Fattahi and Davide Spinello. A timoshenko beam reduced order model for shape tracking with a slender mechanism. *Journal of Sound and Vibration*, 333(20):5165–5180, 2014.

[40] John Till, Vincent Aloi, and Caleb Rucker. Real-time dynamics of soft and continuum robots based on cosserat rod models. *The International Journal of Robotics Research*, 38(6):723–746, 2019.

[41] Albert Edward Green, PM Naghdi, and ML Wenner. On the theory of rods. i. derivations from the three-dimensional equations. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 337(1611):451–483, 1974.

[42] C. Duriez. Control of elastic soft robots based on real-time finite element method. In *2013 IEEE International Conference on Robotics and Automation*, pages 3982–3987, May 2013.

[43] D. C. Rucker and R. J. Webster. Computing jacobians and compliance matrices for externally loaded continuum robots. In *2011 IEEE International Conference on Robotics and Automation*, pages 945–950, May 2011.

[44] Thomas George Thuruthel, Egidio Falotico, Federico Renda, and Cecilia Laschi. Learning dynamic models for open loop predictive control of soft robotic manipulators. *Bioinspiration & biomimetics*, 12(6):066003, 2017.

[45] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[46] Charles W Anderson, Minwoo Lee, and Daniel L Elliott. Faster reinforcement learning after pretraining deep networks to predict state dynamics. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–7. IEEE, 2015.

[47] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

[48] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[49] Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832). Available athttp://underactuated.mit.edu/(2019/06/23).

[50] Brian Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2015.

[51] Kevin M Lynch and Frank C Park. *Modern Robotics*. Cambridge University Press, 2017.

[52] Izrail Moiseevitch Gelfand, Richard A Silverman, et al. *Calculus of variations*. Courier Corporation, 2000.

[53] Vladimir Igorevich Arnol'd. *Mathematical methods of classical mechanics*, volume 60. Springer Science & Business Media, 2013.

[54] Valentin Sonneville, Alberto Cardona, and O Brüls. Geometrically exact beam finite element formulated on the special euclidean group se (3). *Computer Methods in Applied Mechanics and Engineering*, 268:451–474, 2014.

[55] Ed. H. MÃijhlhaus, J. Wiley, N. Y. Ch, and Roderic Lakes. Continuum models for materials with microstructure, 1995.

[56] Juan Carlos Simo and Loc Vu-Quoc. On the dynamics in space of rods undergoing large motionsâĂŤa geometrically exact approach. *Computer methods in applied mechanics and engineering*, 66(2):125–161, 1988.

[57] Carter S Haines, Na Li, Geoffrey M Spinks, Ali E Aliev, Jiangtao Di, and Ray H Baughman. New twist on artificial muscles. *Proceedings of the National Academy of Sciences*, page 201605273, 2016.

[58] Jonghoon Park and Wan-Kyun Chung. Geometric integration on euclidean group with application to articulated multibody systems. *IEEE Transactions on Robotics*, 21(5):850–863, 2005.

[59] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer Science & Business Media, 2006.

[60] Hans Munthe-Kaas. Runge-kutta methods on lie groups. *BIT Numerical Mathematics*, 38(1):92–111, 1998.

[61] Hans Munthe-Kaas. High order runge-kutta methods on manifolds. *Applied Numerical Mathematics*, 29(1):115–127, 1999.

[62] John P Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.

[63] François Demoures, François Gay-Balmaz, Marin Kobilarov, and Tudor S Ratiu. Multisymplectic lie group variational integrator for a geometrically exact beam in r3. *Communications in Nonlinear Science and Numerical Simulation*, 19(10):3492–3512, 2014.

[64] Joachim Linn, Holger Lang, and Andrey Tuganov. Geometrically exact cosserat rods with kelvin–voigt type viscous damping. *Mechanical Sciences*, 4(1):79–96, 2013.

[65] Harvard Lomax, Thomas H Pulliam, and David W Zingg. *Fundamentals of computational fluid dynamics*. Springer Science & Business Media, 2013.

[66] Richard Courant, Kurt Friedrichs, and Hans Lewy. On the partial difference equations of mathematical physics. *IBM journal of Research and Development*, 11(2):215–234, 1967.

[67] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[68] Thomas F Coleman and Yuying Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on optimization*, 6(2):418–445, 1996.

[69] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[70] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[71] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[72] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[73] Matthias Plappert. keras-rl. https://github.com/keras-rl/keras-rl, 2016.

[74] C. Finn, M. Zhang, J. Fu, X. Tan, Z. McCarthy, E. Scharff, and S. Levine. Guided policy search code implementation, 2016. Software available from rll.berkeley.edu/gps.

[75] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Appendix A

Derivation of $t'_{a,i}$

Start with

$$t_{a,i}\|p'_{a,i}\| = p'_{a,i} \tag{A.1}$$

Take the derivative

$$t'_{a,i}\|p'_{a,i}\| + t_{a,i}\|p'_{a,i}\|' = p''_{a,i} \tag{A.2}$$

Now take the cross product between the two equations

$$\hat{p'}_{a,i}p''_{a,i} = \|p'_{a,i}\|^2\hat{t}_{a,i}t'_{a,i} \tag{A.3}$$

Take the cross product again with $t_{a,i}$

$$\hat{t}_{a,i}\hat{p'}_{a,i}p''_{a,i} = \|p'_{a,i}\|^2\hat{t}^2_{a,i}t'_{a,i} \tag{A.4}$$

Using $\hat{a}\hat{b}c = b(a \cdot c) - c(a \cdot b)$, $t_{a,i} \cdot t_{a,i} = 1$, and $t'_{a,i} \cdot t_{a,i} = 0$ we get:

$$\hat{t}^2_{a,i}t'_{a,i} = -t'_{a,i} \tag{A.5}$$

Substituting and rearranging thus gives:

$$t'_{a,i} = -\frac{\hat{p}'^2_{a,i}}{\|p'_{a,i}\|^3}p''_{a,i} \tag{A.6}$$